

版权注意事项：1、书籍版权归著者和出版社所有；  
2、本PDF仅用于个人获取知识，进行私底下知识交流；  
3、PDF获得者不得在互联网以任何目的进行传播；  
如有需要，请尽量购买正版实体书！支持书籍作者！！



作者荣获美国政府颁发的“美国杰出人才”称号。大润发中国区董事长、飞牛网首席执行官黄明端先生与eBay全球零售科学高级总监逄伟先生作序力荐！

将技术与商业需求相结合，深入剖析大数据商业应用中的困惑与难题，帮助读者更好地掌握技术支撑业务高速发展的方案！



技术丛书



Using Big Data to Build Your Business

# 大数据架构商业之路

## 从业务需求到技术方案

黄申◎著



机械工业出版社  
China Machine Press

目前大数据技术已经日趋成熟，但是业界发现与大数据相关的产品设计和研发仍然非常困难，技术、产品和商业的结合度还远远不够。这主要是因为大数据涉及范围广、技术含量高、更新换代快，门槛也比其他大多数IT行业更高。人们要么使用昂贵的商业解决方案，要么花费巨大的精力摸索。本书通过一个虚拟的互联网O2O创业故事，来逐步展开介绍创业各个阶段可能遇到的大数据课题、业务需求，以及相对应的技术方案，甚至是实践解析；让读者身临其境，一起来探寻大数据的奥秘。书中会覆盖较广泛的技术点，并提供相应的背景知识介绍，对于想进一步深入研究细节的读者，也可轻松获得继续阅读的方向和指导性建议。

## 作者简介

---

**黄申** 博士，毕业于上海交通大学计算机科学与工程专业，师从俞勇教授。微软学者，IBM ExtremeBlue天才计划成员。长期专注于



大数据相关的搜索、推荐、广告以及用户精准化领域。曾在微软亚洲研究院、eBay中国、沃尔玛1号店和大润发飞牛网担任要职，带队完成了若干公司级的战略项目。同时著有20多篇国际论文和10多项国际专利，兼任《计算机工程》期刊特邀审稿专家。因其对业界的卓越贡献，2015年获得美国政府颁发的“美国杰出人才”称号。



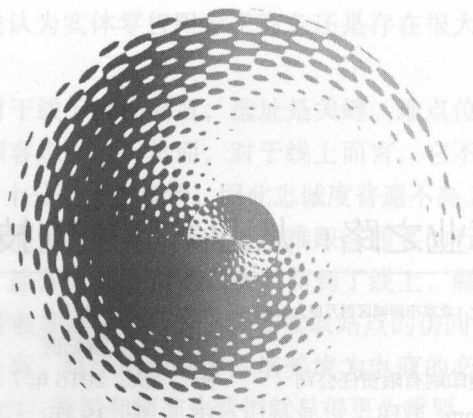
技术丛书

Using Big Data to Build Your Business

# 大数据架构商业之路

## 从业务需求到技术方案

黄申◎著



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

大数据架构商业之路：从业务需求到技术方案 / 黄申著. —北京：机械工业出版社，2016.4  
(2016.7 重印)  
(大数据技术丛书)

ISBN 978-7-111-53528-7

I. 大… II. 黄… III. 商业管理—数据管理 IV. F712

中国版本图书馆 CIP 数据核字 (2016) 第 078609 号

## 大数据架构商业之路：从业务需求到技术方案

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：余 洁

责任校对：董纪丽

印 刷：北京文昌阁彩色印刷有限责任公司

版 次：2016 年 7 月第 1 版第 2 次印刷

开 本：186mm×240mm 1/16

印 张：19.75

书 号：ISBN 978-7-111-53528-7

定 价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东



## Foreword 推荐序一

大润发是1998年成立的，到了2008年已是中国最具规模的大卖场，那时候电子商务刚在萌芽阶段，而实体店也还在快速发展，加上2010年公司忙于筹备上市，准备于2011年在香港挂牌，所以我们并没有花太多时间研究电子商务，而且在那个时间段大部分电子商务公司都处于亏损状态。

后来我们惊觉电子商务已快速发展，办公室很多同事都开始在网上购物了，加上“双11”的天量交易额，逼得我们不得不认真地研究电子商务的发展趋势。到2012年，我们发现电子商务越做越好，尤其进入移动互联网时代后，想要满足顾客随时随地的购物需求，电商发展必然是未来消费的新渠道与趋势。所以我们在2013年决定进军电子商务并成立飞牛网(Feiniu.com)。

经过两年多的实践，我认为实体零售跟电子商务还是存在很大差异，其中最主要的差异有两点。

- 顾客忠诚度差异：对于线下卖场而言，选址是关键。地点位置正确，就会有稳定的客流，也容易培养顾客忠诚度。然而，对于线上而言，它不受地域的限制，顾客切换不同的网站是一件十分轻松的事情，因此忠诚度普遍不高。
- 顾客行为数据的获取成本差异：线下卖场很难跟踪顾客的行为，如果要安装各种复杂的信息采集设备，运营成本就会很昂贵。而到了线上，顾客浏览网站时“凡走过必留下痕迹”，电商要收集顾客的行为只需要读取站点的访问日志，可以说相对容易。正是因为顾客忠诚度不高，对于忠诚顾客的培养成为电商的必争之地。我相信，要实现这个目标，基于顾客行为的大数据和精准化营销就显得更为重要。我们需要充分利用数据挖掘，并快速反馈到整个电商系统。

至于如何做到个性化的搜索和推荐，如何做好客户关系管理(CRM)，以及如何做到精准的推送和营销，一直是我们探索的内容。飞牛网从成立之初到现在，碰到了很多与搜索和大数据相关的问题和困难，一年半以前，黄申博士加入了飞牛网的技术团队，他的技术和经验对于我们的帮助很大，在他的指导下我们快速建立了专业的搜索、推荐及用户画像系统。这些都是我们分析顾客、理解顾客、提升顾客在线体验的核心，使得飞牛网和行业先锋之间

的距离在短时间内大幅缩小。关注飞牛的读者，你们可以到飞牛网体验一下个人喜欢的商品，然后你就能细细品味到我们搜索、推荐等大数据相关的功能给你带来的便捷和惊喜。

当然，这些成绩和黄申博士丰富的业界经验分不开。在日常的工作中，他总是有独到的见解。如果你有幸阅读本书，一定能从他的分享中了解大数据是如何运作的，了解大数据是如何支持业务的，以及了解技术是如何满足业务需求的。对于还处在大数据摸索中的人而言，他的思路和探讨非常宝贵，这是一本讲述搜索和大数据领域实践经验的好书，值得推荐。

飞牛网 CEO 黄明端

2016年3月

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

本书在出版前，曾得到许多业内人士的支持和鼓励。在本书出版之际，我要向所有支持本书出版的朋友们致以诚挚的谢意。同时，我也要感谢我的家人，是他们在我遇到困难时给予我无私的支持和鼓励。最后，我要感谢我的同事，是他们在我工作之余，抽出宝贵的时间，帮助我完成了这本书的编写工作。

## Foreword 推荐序二

最近的十年中，我一直在 eBay 从事数据相关的项目，领导了包括零售科学和搜索科学在内的研发团队。如今 eBay 在全球已有 1 亿多注册用户，每天都有数以百万的家具、收藏品、电脑、车辆在 eBay 上被刊登、贩售、卖出，每年的营业额高达数千亿美元。

但是我们非常清楚，对于 eBay 而言，更为珍贵的财富是网站上每时每刻都会产生的海量数据。通过对这些数据的分析，我们可以指导卖家进行更好的搜索引擎优化、制定更好的价格、控制合理的库存；还可以帮助买家找到更合适自己的商品和更优质的服务。当然，大数据的分析还能帮助 eBay 有效地防范作弊和欺诈，保证整个平台和渠道的健康发展。

正是意识到数据的关键性，eBay 非常重视挖掘和利用它们的潜在价值。本书作者曾经在 eBay 的研究院和搜索科学部门工作，专门从事机器学习的研究和应用。他协助 eBay 构建了数项核心算法及其相关产品，包括基于机器学习的搜索排序、高质量用户评价的发现和摘要、相似和相关商品推荐栏位等。在此过程中，他和各个技术同仁、产品经理、业务部门紧密合作，而这本著作就融入了作者在这些实战项目中所积累的丰富经验。所以，本书最大的闪光点在于，它的内容不仅仅局限于技术本身，而是考虑到了在不同的应用场景下，这些技术应该怎样合理运用。例如，对于基于学习的搜索排序，通常要考虑哪些因素以及怎样的学习模型？对于智能推荐的栏位而言，相似和相关商品又有怎样的区别？分别都应该使用怎样的推荐模型？

除了与业务应用紧密结合，此书还具有覆盖面广和通俗易懂的特点。全书涉及的主题包括大数据的获取、存取、处理、检索、挖掘和评估中的多数主流技术。同时，作者从自己独特的视角出发，对深奥的技术进行了深入浅出的阐述，大幅降低了大数据知识理解的难度。因此，本书也非常适合大数据产品设计者、产品经理或者架构师进行阅读。我相信，对于希望利用大数据解决业务痛点的读者而言，此书是不可或缺的良好良师益友。

eBay 全球高级总监 逢伟

2016 年 3 月

## 前言 Preface

### 为什么要写这本书

李克强总理提出“大众创业，万众创新”。在如此美好的大环境下，互联网创业如火如荼。各种模式的 O2O，各种精彩的移动 App，突然之间都冒了出来，正所谓“忽如一夜春风来，千树万树梨花开”。而在其中，大数据因为蕴含着巨大的商业价值，成为这个时代的趋势之一。众人都希望利用好这个“魔棒”，为自己的事业开疆扩土。可是，就笔者在业界的经历来看，真正能挖掘大数据潜力的公司少之又少。笔者一直很好奇，中国的相关人才如此之多，商业市场又如此之大，何以至如此境地呢？为了找到答案，笔者阅读了不少观察性文章，也走访了一些业内的从业者，发现目前的一大窘境是：大数据技术、产品和商业的结合度还远远不够。导致这个现状的原因有很多，具体分析主要有以下几点：

- 涉及范围广：“大数据”本身是一个比较抽象的概念，任何关乎大规模数据的处理，都可以称为“大数据”。因此它既包括了很多已有的技术，如数据挖掘、机器学习、商业智能等，又包括了近几年诞生的新技术<sup>①</sup>，如 NoSQL 相关的生态系统。而且，一个商业需求也可能会涉及多个相关技术。
- 技术含量高：数据挖掘和机器学习之类的算法和大规模数据处理的架构，相对于普通的应用开发而言，需要更多的理论知识和实践经验积累。而商业价值的挖掘程度却往往取决于使用的技术深度。越是钻研得深入，所产生的价值就会越大。
- 发展速度快：最近几年，算法方面有不少的创新，如深度学习（Deep Learning）；系统架构也在不断升级，如 Hadoop 的第二代框架 Yarn、Storm、Spark 等实时流式计算，技术的更新换代非常频繁。但是，商业的发展需要技术系统能够随时应变，快速响应，这与技术的飞速发展本身又存在冲突。
- 成熟方案少：大数据的技术多数是免费的，这对于盈利模式而言无疑是有利的，不过代价就是存在一定的稳定性和易用性问题。现在有一些大型的技术公司提供了更

① 请注意，这里的“新”是相对的，互联网和 IT 技术发展之快，令人咋舌。



成熟的解决方案，但是价格不菲，对于经费并不宽裕的初创公司而言选择余地太少。

以上这些因素都会形成进入大数据领域的门槛，而高门槛势必会导致大数据在工业界应用的步伐放缓。为了解决这个问题，企业需要培养自己的复合型人才，要求业务人员懂技术、技术人员懂业务。只有如此才能让公司使用合适的工具、获得准确的数据、制定合理的方案。

然而，激烈的市场竞争，膨胀的用户需求，不会给创业公司太多的时间去挥霍。在黑夜之中不断摸索的人们，需要明灯指引前进的方向。虽然目前市面上已有一些相关图书做了不错的尝试，但是它们大多数偏向两个极端：一端是面向金融、经济、社会和管理类等非技术型读者，讲述概念、定义、背景和业界的成功案例等；另一端是面向程序员、算法工程师、架构师和数据科学家等纯技术型读者，讲述具体的技术框架、编程范例、系统调试等。能同时覆盖两者的图书可谓凤毛麟角。因此，笔者萌生了通过一本书来帮助企业快速地建立复合型团队，将合理的业务需求尽快转化为实际产品的想法。笔者在写作过程中，力求：

- 易懂易懂。通过生动的案例和形象的比喻来解读难点，降低技术理解的门槛。这样就能够让偏向业务的人员更容易理解大数据背后的运作原理，促进他们和技术人员的沟通及协作。

- 可实践性强。通过分享需要大量实践才能积累的宝贵经验，最大程度地针对业务需求和技术方案之间的空白进行弥补。这将有利于技术人员针对不同的业务需求，规划更为合理的技术方案。

本书通过讲述一个虚拟的（如有雷同纯属巧合）互联网 O2O 创业故事，逐步展开介绍各个阶段可能遇到的大数据课题、业务需求，以及相对应的技术方案，甚至是实践解析。让读者身临其境，一起来探寻大数据的奥秘。对于想进一步深入研究技术实现细节的读者，也给出了继续阅读的方向和指导性建议。笔者衷心希望，无论是技术专家、产品经理，还是业务人员，只要阅读了本书便都能愉快地遨游在大数据的海洋中。

## 读者对象

根据本书撰写的起心动念，笔者觉得其内容适合如下读者：

- 中小互联网创业公司的 CIO、CTO 和技术骨干。他们可以获知常见的互联网公司从创业初期到中期这个阶段里，数据平台需要满足怎样的业务需求（当然，也包括业务方和产品经理所说的“XXOO”了），技术上通常会面临哪些挑战，以及如何解决。

- 中小互联网创业公司的产品经理和项目经理。个人认为，在不久的将来，最炙手可热的产品经理或项目经理一定是懂一些技术的。技术背景将帮助产品经理和项目经理更好地理解哪些是技术上可以实现的，如果可以实现又大致需要多少开发资源。此外，本书所提及的案例也许能提供一些产品设计上的灵感和启发。

- 中小互联网创业公司的 CEO、合伙人。读懂这本书，CIO、CTO 和产品 VP 的招募，

不用靠第三方和人力资源，因为你可以自己来选。这绝对可以帮助公司少走弯路，加速发展。

□ 刚刚起步的算法和架构工程师。很多刚刚毕业或工作没多年的朋友，学了一身本领，对新技术也很有热情，苦于没有太多实践的机会。书中的故事浓缩了不少业界实践的经验 and 心得，如能融会贯通对他们将很有裨益。同时，覆盖面较广的技术课题概述也为他们继续深入研究提供了方向和指导。

□ 梦想家。最后的最后，本书也献给那些希望通过大数据技术进行互联网创业的人们。也许现在你既不是“CXO”（CEO、CIO、CTO、CPO、COO 等的统称），也不是产品经理或项目经理，可是你有自己的创业梦想，那么这本书也献给你。

当然，由于侧重点不同，因此本书并不适合钻研技术细节的程序员和编程专家，不过仍然可以在书中找到重要的参考图书指导。同时，本书也不适合关注宏观行业发展的商务人士。

## 如何阅读本书

为了达到深入浅出、通俗易懂的效果，本书的第一大部分概述了大数据的主要技术，包括大数据的获取、存储、处理，还有架构设计的基本理念，以及常用的消息和缓存机制。这一部分你会发现关于 Nutch、Flume、Hadoop、HBase、Redis、Hive、Kafka、Spark、Storm 等的简介。对于数据处理的高级技术，本书着墨不少，但不乏对于信息检索和数据挖掘课题的探讨。例如站内搜索引擎、推荐系统、广告系统、聚类、分类和线性回归等。由于商业需求尤其看重实际产出，因此第一部分的最后还会分析常见的效果和性能评估。相信这部分对于构建读者的大数据知识体系会很有帮助。在每一章的最后，我们还会给出重要的参考图书，以便于读者继续深入学习。

第二大部分的每个章节都是从业务需求的描述入手，然后进行需求分析，根据需求的特点，对第一大部分所涉及的备选技术进行筛选，最后是技术方案和架构的确定。不同的商业需求可能会使用类似的技术点。但是具体使用方式不会雷同，根据不同的数据集合、不同的应用场景和不同的进阶难度，我们为读者提供了反复温习和加深印象的机会。

## 勘误和支持

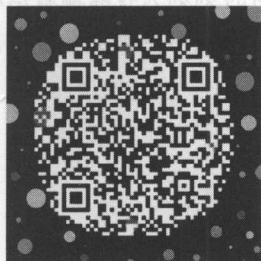
正如前文所述，大数据发展得实在是太快了。可能就在你阅读这段文字的同时，又有一项新的技术诞生了，N 项技术升级了，M 项技术被淘汰了。再加之笔者的水平有限，编写的时间也较仓促，书中难免会出现一些不够准确或有遗漏的地方，不妥之处在所难免，恳请读者通过如下渠道积极建议和斧正，我们很期待能够听到你们的真挚反馈。

QQ: 36638279

微信: 18616692855

邮箱: s\_huang790228@hotmail.com

LinkedIn: <https://cn.linkedin.com/in/shuang790228>



扫一扫就能联系作者:



公众号:

## 致谢

首先要感谢上海交通大学尤其是俞勇教授,你们给予我不断学习的机会,带领我进入了大数据的世界。同时,感谢阿里云的高级总监薛贵荣,你的指导让我树立了良好的科研态度。

还要感谢微软亚洲研究院、eBay 中国研发中心、沃尔玛 1 号店、大润发飞牛网和 IBM 中国研发中心,在这些公司十多年的实战经验让我收获颇丰,也为本书的铸就打下了坚实的基础。

感谢曾经的微软战友陈正、孙建涛、Ling Bao、曾华军、张本宇、沈抖、刘宁、严峻、曹云波、王琼华、康亚滨、胡健、季蕾等, eBay 的战友逢伟、王强、王骁、沈丹、Yongzheng Zhang、Catherine Baudin、Alvaro Bolivar、Xiaodi Zhang、吴晓元、周洋、胡文彦、宋荣、刘文、Lily Yu 等, 沃尔玛 1 号店的战友韩军、王欣磊、胡茂华、付艳超、张旭强、黄哲铿、沙燕霖、郭占星、聂巍、邵汉成、张珺、胡毅、邱仔松、孙灵飞、凌昱、王善良、廖川、杨平、余迁、周航、吴敏、李峰等, 大润发飞牛网的战友王俊杰、陈俞安、蔡伯璟、陈慧文、夏吉吉、文燕军、杨立生、张飞、代伟、陈静、赵瑜、李航等, IBM 的战友李伟、谢欣、周健、马坚、刘钧、唐显莉等。要感谢的同仁太多,如有遗漏敬请谅解,很怀念和你们并肩作战的日子,你们让我学到了很多。

感谢机械工业出版社华章公司的编辑杨绣国(Lisa)老师,感谢你的魄力和远见,在最近的 3 个月中始终支持我的写作,你的鼓励和帮助引导我顺利地完成了全部书稿。也要感谢凌云为我引荐了如此优秀的出版社和编辑。

衷心感谢大润发、飞牛网董事长黄明端先生和 eBay 全球高级总监逢伟先生,在百忙之中为本书作序。也衷心感谢欧电云的董事长韩军先生、永辉集团电商总经理黄志雄先生、美的集团电商总经理吴海泉先生、百度 LBS 新业务产品总监王欣磊先生、阿里巴巴高级产品专家张旭强先生、LinkedIn(领英)的商务分析经理 Yongzheng Zhang 先生、京东商城推荐





## Contents 目 录

### 推荐序一

### 推荐序二

### 前 言

## 第1章 抉择..... 1

## 第2章 数据收集..... 4

### 2.1 互联网数据收集..... 4

#### 2.1.1 网络爬虫..... 5

#### 2.1.2 Apache Nutch 简介..... 11

#### 2.1.3 Heritrix 简介..... 14

### 2.2 内部数据收集..... 15

#### 2.2.1 Apache Flume 简介..... 17

#### 2.2.2 Facebook Scribe 和

#### Logstash..... 21

### 2.3 本章心得..... 21

### 2.4 参考资料..... 22

## 第3章 数据存储..... 23

### 3.1 持久化存储..... 23

#### 3.1.1 Hadoop 和 HDFS..... 25

#### 3.1.2 HBase 简介..... 28

#### 3.1.3 MongoDB..... 35

### 3.2 非持久化存储..... 37

#### 3.2.1 缓存和散列..... 37

#### 3.2.2 Memcached 和 Berkeley

#### DB 简介..... 41

#### 3.2.3 Redis 简介..... 41

### 3.3 本章心得..... 44

### 3.4 参考资料..... 44

## 第4章 数据处理..... 46

### 4.1 离线批量处理..... 46

#### 4.1.1 Hadoop 的 MapReduce..... 47

#### 4.1.2 Spark 简介..... 52

#### 4.1.3 Hive 简介..... 53

#### 4.1.4 Pig、Impala 和 Spark SQL..... 56

### 4.2 提升及时性：消息机制..... 58

#### 4.2.1 ActiveMQ 简介..... 60

#### 4.2.2 Kafka 简介..... 61

### 4.3 在线实时处理..... 63

#### 4.3.1 Storm 简介..... 63

#### 4.3.2 Spark Streaming 简介..... 66

### 4.4 本章心得..... 66

### 4.5 参考资料..... 67

## 第5章 信息检索..... 69

### 5.1 基本概念..... 70

5.2 相关性.....	70	6.3.1 监督学习——分类.....	137
5.2.1 布尔模型.....	70	6.3.2 监督学习——回归.....	152
5.2.2 基于排序的布尔模型.....	71	6.3.3 非监督学习——聚类.....	153
5.2.3 向量空间模型.....	74	6.4 挖掘工具.....	157
5.2.4 语言模型.....	75	6.4.1 Mahout 简介.....	157
5.3 及时性.....	77	6.4.2 R 简介.....	159
5.4 与数据库查询的对比.....	81	6.5 本章心得.....	165
5.5 搜索引擎.....	82	6.6 参考资料.....	165
5.5.1 Web 搜索中的链接分析.....	83		
5.5.2 电子商务中的商品排序.....	86	<b>第 7 章 效能评估</b> .....	167
5.5.3 多因素和基于学习的排序.....	88	7.1 效果评估.....	168
5.5.4 系统框架.....	89	7.1.1 离线评估.....	169
5.5.5 Lucene 简介.....	93	7.1.2 非离线的评估.....	183
5.5.6 Solr 简介.....	98	7.2 性能评估.....	190
5.5.7 Elasticsearch 简介.....	104	7.2.1 计算复杂度.....	191
5.6 推荐系统.....	108	7.2.2 应用系统性能.....	193
5.6.1 推荐的核心要素.....	109	7.2.3 JMeter 工具.....	197
5.6.2 推荐系统的分类.....	110	7.3 本章心得.....	202
5.6.3 混合模型.....	115	7.4 参考资料.....	202
5.6.4 系统架构.....	116		
5.6.5 Mahout.....	116	<b>第 8 章 大数据技术全景</b> .....	204
5.7 在线广告.....	119		
5.7.1 在线广告的类型.....	120	<b>第 9 章 商品太多啦! 需要搜索引擎</b> .....	207
5.7.2 广告投放机制.....	124	9.1 业务需求.....	207
5.7.3 广告的拍卖机制.....	125	9.2 产品设计和技术选型.....	208
5.7.4 广告系统架构.....	126	9.3 实现方案.....	211
5.8 本章心得.....	127	9.3.1 数据定义和配置.....	211
5.9 参考资料.....	128	9.3.2 集群搭建.....	213
		9.3.3 DIH 配置.....	216
<b>第 6 章 数据挖掘</b> .....	130		
6.1 基本概念.....	131	<b>第 10 章 能否更主动? 还需要推荐引擎</b> .....	223
6.2 数据的表示和预处理.....	133	10.1 业务需求.....	223
6.2.1 数据的表示.....	133	10.2 产品设计和技术选型.....	225
6.2.2 数据的预处理.....	135		
6.3 机器学习算法.....	136		

10.3 实现方案.....	230	12.6 “还要搜得更准”的方案实现.....	271
10.3.1 基于内容特征的衡量.....	230	12.7 业务需求：还要更快.....	273
10.3.2 基于行为特征的衡量.....	233	12.8 还要“变”得更快：产品设计 和技术选型.....	274
10.3.3 提供在线服务.....	236	12.9 还要“搜”得更快：产品设计 和技术选型.....	275
<b>第 11 章 这样做的效果如何.....</b>	<b>241</b>	12.10 业务需求：给点提示吧.....	280
11.1 业务需求.....	241	12.11 给点提示吧：产品设计和 技术选型.....	282
11.2 产品设计和技术选型.....	242	<b>第 13 章 支持更高效的运营.....</b>	<b>287</b>
11.3 实现方案.....	243	13.1 业务需求：互联网时代的 CRM.....	287
11.3.1 行为数据的定义和记录.....	243	13.2 互联网时代的 CRM：产品 设计和技术选型.....	288
11.3.2 Flume 和 HDFS 的集成.....	246	13.3 业务需求：抓住捣蛋鬼.....	291
11.3.3 通过 Hive 进行分析.....	252	13.4 抓住捣蛋鬼：产品设计和 技术选型.....	292
11.3.4 Kafka 和 Storm 的集成.....	254	13.4.1 识别分类错放.....	292
<b>第 12 章 这个搜索有点逊.....</b>	<b>258</b>	13.4.2 识别 SEO 作弊.....	294
12.1 业务需求：还要搜得更多.....	258	13.5 业务需求：销售之战.....	295
12.2 “还要搜得更多”：产品设计 和技术选型.....	259	13.6 销售之战：产品设计和技 术选型.....	296
12.3 “还要搜得更多”的方案实现.....	261	13.6.1 设置合理的价格.....	296
12.3.1 HBase 的部署.....	261	13.6.2 识别黄牛.....	298
12.3.2 HBase 和 Solr 的集成.....	264	<b>后记.....</b>	<b>299</b>
12.4 业务需求：还要搜得更准.....	265		
12.5 “还要搜得更准”：产品设计 和技术选型.....	266		
12.5.1 提升搜索排序的相关性.....	266		
12.5.2 提升搜索排序的整体 效果.....	268		

## 第1章

## Chapter 1

# 抉 择

上海，又是一个春天，阳光透过薄薄的窗帘，懒懒散散地洒入屋内。当一缕光线偷偷地爬上杨大宝的眼角时，他睁开了朦胧的双眼。

等等！杨大宝是何许人也？

杨大宝，姓杨名大宝，土生土长的上海人，从小就喜欢玩电子产品，大学专业是计算机科学，酷爱信息技术和互联网。自从大学毕业后，他就一直任职于一家大的IT公司。最近，他面临人生的一项重大选择。原来，有几位志同道合的朋友想拉他一起开创自己的公司。大宝很清楚，这几年中国迎来了创业的黄金时代。李克强总理提出的“大众创业，万众创新”，明确了政策对创业的大力支持。而老百姓的生活水平也在不断提高，各方面的需求也在不断增加，同时各种风险投资也非常充裕。在这样的大背景下，大家的创业热情空前高涨，尤其是互联网，简直可以用“疯狂”来形容。大宝觉得这正是一个实现自己梦想的好契机。不过，放弃目前优厚的薪资待遇和受人尊敬的公司职位，和几个小伙伴去闯荡江湖，也是要冒不少风险的，最终是否能成功也充满了变数，这样做到底值得吗？大宝这几天夜不能寐，晚上做梦也要纠结一番。若不是淘气的阳光溜进来，可能他还要继续在梦里思考。

洗漱完毕，大宝一边吃着早餐，一边接着梳理思路。首先，创业的点子是不错的，主要思想是做线上线下O2O（Offline to Online）的社区商业模式：将大型社区周边的各种服务行业进行线上化，让用户足不出户，就可以叫外卖、订座、享受美甲、按摩等服务，还可以购买商品。用户的生活需求得到更大程度的满足，商家也可以吸引到更多的线上客流，而公司的平台也能从双方的交易中获得收益，形成多方互赢的局势，市场前景一片光明。其次，因为大宝是团队里唯一懂IT技术的骨干，那么公司里整个庞大的网络系统架构肯定会由他来负责。这几年的工作经历让他也积攒了不少设计和开发的实战经验。后端如数据库、ERP（Enterprise Resource Planning）系统、图片服务器，前端如会员注册、购物流程、页面展示



等大宝都有很深入的了解。不过他还是隐约觉得缺了些什么。

吃完了早餐，大宝熟练地打开电脑，开始飞快地在网上查阅资料，钻研成功的互联网站点是如何设计和架构的。就这样，时钟滴滴答答，不知不觉一天过去了。随着夜幕的降临，望着窗外柔和的街灯，大宝深深地吐了一口气，“还缺一个关键词：大数据”，这是他一天研究下来的结论。

等等？大数据又是什么？

好问题，其实此刻大宝心里也没谱，但是他看到好多资料都反复提到这个词。他隐约觉得，如果没有摸清这点，对于这个初创公司而言，就会存在很大的不确定性。可是，目前创业的团队也很多，竞争相当激烈，从来都不缺好的创意，就看谁首先能做得出、做得好、做得快。没有太多的时间留给大宝了。那该如何是好呢？突然，大宝想到一个人，也许能为他解决心中的这个疑惑。

此人就是黄小明，是大宝的表哥。他是知青子女，从小随父母到武汉生活和读书，到16岁的时候回到上海，考入了知名的高校，并且获得了计算机科学的博士学位，可谓知识渊博。毕业后他在几家世界知名的互联网和电子商务公司任职，有十多年的科研和开发经验，目前正在带领团队攻关几个核心项目。

终于，在一个美好的周末下午茶时间，大宝约到了小明。大宝开门见山，针对自己目前的状况和思考的问题进行了说明。

“嗯……大宝，大数据的确是一个非常重要的领域，而且想要上手也有一定的难度。”

“哦，为什么呢？”

“大数据入门的门槛比较高，原因有几点：知识面非常广，技术含量也比较高，此外发展和更新的速度也快得惊人。更为关键的是，这些技术一般都是开源的，很多都需要自己摸索和积累。除非你们考虑直接使用一些大公司比较成熟的付费方案。”

“嗯，如果是创业起步阶段，我们肯定是不考虑昂贵的商业解决方案的。”

“那问题就更复杂了……不过……”

“不过什么？”

“如果你肯花些功夫来学习，或许我能给你一些建议和启发。”

“哈哈，小明哥，搞了半天你是要自卖自夸啊！”

“这都被你看出来了。其实我最近正在整理这些年的心得体会，准备出版一本关于大数据的书，以便于团队的培养及业界的交流。那我借此机会，先和你讲讲，如何？”

“哇，那求之不得啊！”

“大数据其实是非常宽泛的概念，这里我强调的是如何获取海量的数据，并对它们进行有效的存储、处理和分析，最终让其服务于我们的业务需求。首先，要知道数据的来源非常关键，没有数据就没有生产的原材料。所以我考虑先阐述什么是站外和站内的数据收集系统，以及哪些开源工具可以帮助我们。对于收集到的数据，在第一时间我们要存储它们，然后介绍最近流行的分布式存储系统，确保辛辛苦苦采集而来的数据不会丢失。并说明对于数

据, 可以进行哪些基本的处理, 以便产生我们所期望的一些数字统计、内容转换等结果。当然, 还有很多高级的技术能够让数据产生更大的价值, 如信息检索和数据挖掘。接着就是信息检索领域了, 包括搜索、推荐、广告等应用。对了, 数据挖掘的概念、基本流程和机器学习的主要算法也很重要。有了这些基础之后, 还要考虑算法、模型等处理的效果和性能问题, 衡量其是否能达到设计的预期。最后将上述知识点串起来, 给出全局的概览框架。你看, 这样的逻辑顺序你能理解吗?”

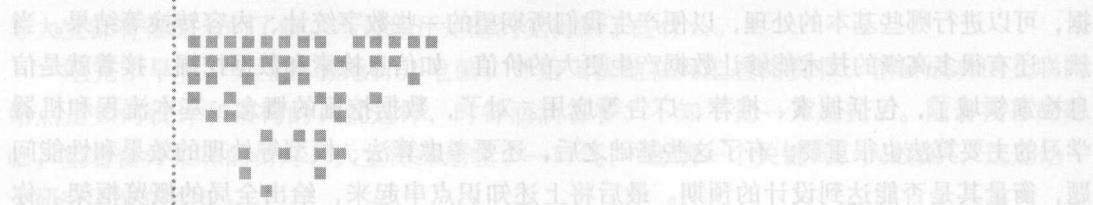
“其实, 都不太懂……你还是现在就开始教我吧, 从你刚说的最基础的开始吧!”

与之非常相似的搜索工具, 并增加了对网页的检索支持。这些搜索工具都离不开收集文件内容的重要模块, 在工作的时候它就像蜘蛛一样在网络间爬来爬去, 因此人们称为“蜘蛛”程序, 也可认为它们是爬虫程序最早的实现。在互联网时代, “蜘蛛”程序进一步演变成成为人们俗称的爬虫模块, 正是利用了爬虫的强大功能, 日益庞大的互联网内容才能突破网络的黑暗状态, 变成一个可知的世界。在不远的未来, 互联网信息的抓取、挖掘和再处理, 将成为更多人的需求, 而满足这种需求的, 就是各种各样的爬虫和与之相关的信息处理工具了。现在网络上流行的信息采集工具、网站爬取工具, 都是未来新一代爬虫的先驱, 甚至已经具备强大的功能, 可以实现从互联网中抓取大量的数据, 并进行各种复杂的分析和处理, 为各种应用提供强大的支持。

爬虫程序是互联网时代的一个重要组成部分, 它的主要作用是自动地从互联网上抓取数据, 并将这些数据存储在本地数据库中。爬虫程序的工作原理非常简单, 它通过模拟浏览器的行为, 向目标网站发送请求, 并接收返回的网页内容。然后, 爬虫程序会对返回的内容进行解析, 提取出其中的链接, 并继续向这些链接发送请求, 从而实现对整个网站的遍历。爬虫程序的应用非常广泛, 可以用于搜索引擎、数据挖掘、内容分发网络等多种场景。在搜索引擎中, 爬虫程序负责抓取网页内容, 并将其索引到搜索引擎的数据库中, 以便用户能够快速找到所需的信息。在数据挖掘中, 爬虫程序可以用于抓取大量的数据, 并进行各种复杂的分析和处理, 以发现其中的规律和趋势。在内容分发网络中, 爬虫程序可以用于抓取网页内容, 并将其缓存到本地的服务器上, 以便用户能够快速访问。

## 1.1 爬虫程序概述

爬虫程序是互联网时代的一个重要组成部分, 它的主要作用是自动地从互联网上抓取数据, 并将这些数据存储在本地数据库中。爬虫程序的工作原理非常简单, 它通过模拟浏览器的行为, 向目标网站发送请求, 并接收返回的网页内容。然后, 爬虫程序会对返回的内容进行解析, 提取出其中的链接, 并继续向这些链接发送请求, 从而实现对整个网站的遍历。爬虫程序的应用非常广泛, 可以用于搜索引擎、数据挖掘、内容分发网络等多种场景。在搜索引擎中, 爬虫程序负责抓取网页内容, 并将其索引到搜索引擎的数据库中, 以便用户能够快速找到所需的信息。在数据挖掘中, 爬虫程序可以用于抓取大量的数据, 并进行各种复杂的分析和处理, 以发现其中的规律和趋势。在内容分发网络中, 爬虫程序可以用于抓取网页内容, 并将其缓存到本地的服务器上, 以便用户能够快速访问。



## Chapter 2

## 第2章

# 数据收集

觉得，如果没有投资这点，对于这个初创公司而言，就会存在很大的不确定性。可是，目前创业的团队也很多，竞争相当激烈，从来都不缺好的创意，就看谁先能做得出，做得好，做得快。没有太多的时间留给大宝了。那该如何是好呢？突然，大宝想到一个人，也许能为他解决心中的这个疑惑。

此人就是黄小明，是大宝的表哥。他是知青子女，从小随父母到武汉生活和读书，到16岁的时候回到上海，考入了知名的高校，并且获得了计算机科学的博士学位，可谓知识渊博。毕业后他在几家世界知名的互联网和电子商务公司任职，有十多年的科研和开发经验。

“小明哥，听上去大数据涵盖的课题非常丰富啊，我之前一点经验都没有，学习起来是不是颇有难度？”

“没关系，我们一步一步地来，先从最基础的数据收集谈起吧。”

大数据这个概念的含义很广，其中最首要的一点就是大规模的数据，这也是目前发展的趋势所在。想想看，十多年前电脑刚刚开始普及的时候，如果谁拥有一个2 GB的硬盘那都是非常值得炫耀的事情了，而如今，即使有一个2 TB的硬盘（相当于1 024个2 GB硬盘），你也经常会发现无法放下所有的个人资料。个人数据尚且如此，更不要提整个信息世界了，它已经完完全全进入了数据爆炸的时代。然而，在拥有大数据的同时，我们又会发现有些数据并非想象的那样唾手可得。例如，互联网上关于某个产品的评论有哪些？在购物网站上，用户们的浏览和购买行为又是怎样的？等等。正所谓“乱花渐欲迷人眼”，面对纷繁复杂的数据我们有时反而感觉无从下手。无法获得所需的数据，我们纵使有再好的技术也无法提炼出任何价值。因此，我们需要通过各种方案，将所有富含潜在价值的数据统统都收集起来，而这个规模可能还不小。

本章首先介绍在互联网获取数据的强大工具——网络爬虫，包括它的工作原理、操作流程、主要类型和相应的开源工具。然后介绍企业内部获取数据的主要思路、流程和相应的开源工具。

“大数据其实是非常宽泛的概念，这里我强调的是如何获取海量的数据，并对它们进行

## 2.1 互联网数据收集

如今的互联网是一个神奇的世界，充满了各种各样的信息。通过像Google、百度、Bing这种世界级的Web搜索引擎，你几乎可以找到任何你想要的答案。让人觉得不可思议

的是,搜索引擎好像能够触达信息世界的每一个角落,发现并收集相关的内容。那么,它们是凭借什么样的神器来达到这个目标的呢?这就是网络爬虫(Web Crawler)。网络爬虫又可称为网络蜘蛛(Web Spider)、网络机器人等,即按照一定的规则,自动地抓取互联网信息的一种程序或脚本。可以认为现代搜索引擎的祖先是1990年由蒙特利尔大学的学生 Alan Emtage 发明的 Archie。Archie 的工作原理与现在的搜索引擎很接近,它依靠脚本程序自动搜索网上的文件,并对有关信息进行索引,之后使用者就可以采用一定的表达式进行查询。受 Archie 的启发,美国内华达 System Computing Services 大学于1993年开发了另一个与之非常相似的搜索工具,并增加了对网页的检索支持。这些搜索工具都离不开收集文件内容的重要模块,在工作的时候它就像蜘蛛一样在网络间爬来爬去,因此人们称为“蜘蛛”程序。也可认为它们是爬虫程序最早的实现。在互联网时代,“蜘蛛”程序进一步演变成为人们俗称的爬虫模块。正是利用了爬虫的强大功能,日益庞大的互联网内容才能突破网络的黑暗状态,变成一个可知的世界。在不远的未来,互联网信息的抓取、挖掘和再处理,将成为更多人的需求,而满足这种需求的,就是各种各样的爬虫和与之相关的信息处理工具了。现在网络上流行的信息采集工具、网站聚合工具,都将是未来新一代爬虫的先驱,甚至已经具备了其特点。另外,随着 Web 2.0 的流行,如何抓取动态页面也成为搜索引擎爬虫需要解决的热点。

## 2.1.1 网络爬虫

首先来看一下网络爬虫的具体工作模式和分类。其主要的工作流程是从一个或若干个初始网页的地址 URL (Uniform Resource Locator) 开始,获得初始网页上的 URL 列表。然后在抓取网页的过程中,不断地从当前页面上抽取新的 URL 放入待爬行队列,直到满足系统的停止条件为止。从这个流程中可以看出,如何有效地获取新的 URL,对于爬虫执行成功与否非常关键,从这个角度可以将网页获取的策略分为几个大类:深度优先、宽度优先和最佳优先三种。

### 1. 深度优先

该策略从起始网页开始,选择一个 URL 进入,分析这个网页中的 URL,然后再选择另一个 URL 进入。如此一个链接一个链接地抓取下去,直到处理完整条路线之后再处理下一条路线,图 2-1 显示了获取的先后顺序,其中每个节点代表一个网页,节点中的数字标号表示该网页的唯一 ID,节点旁边的数字标号代表该网页被访问的顺序,每条实线代表网页之间的链接,每条虚线代表访问的顺序。对于熟悉计算机数据结构的读者而言,深度优先的抓取策略可以用栈(Stack)来实现,步骤如下:

- 1) 将初始网页节点压入栈中。
- 2) 每次从栈中弹出一个节点,搜索在它下一级的所有节点。
- 3) 将第 2 步中新发现的节点压入栈中。



4) 重复第 2 步和第 3 步，直到没有发现新的网页节点为止。  
栈的处理过程参见图 2-2。

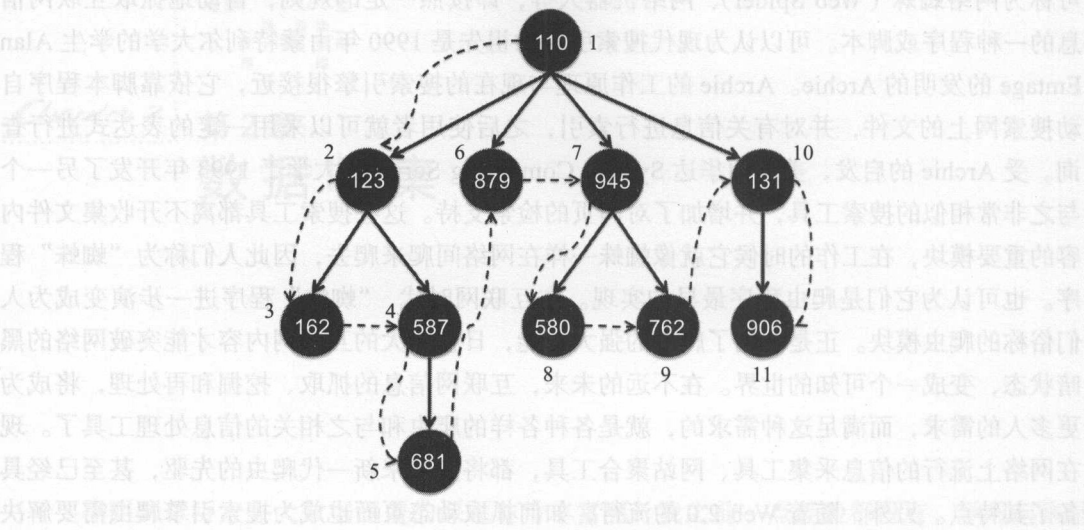


图 2-1 深度优先的获取策略

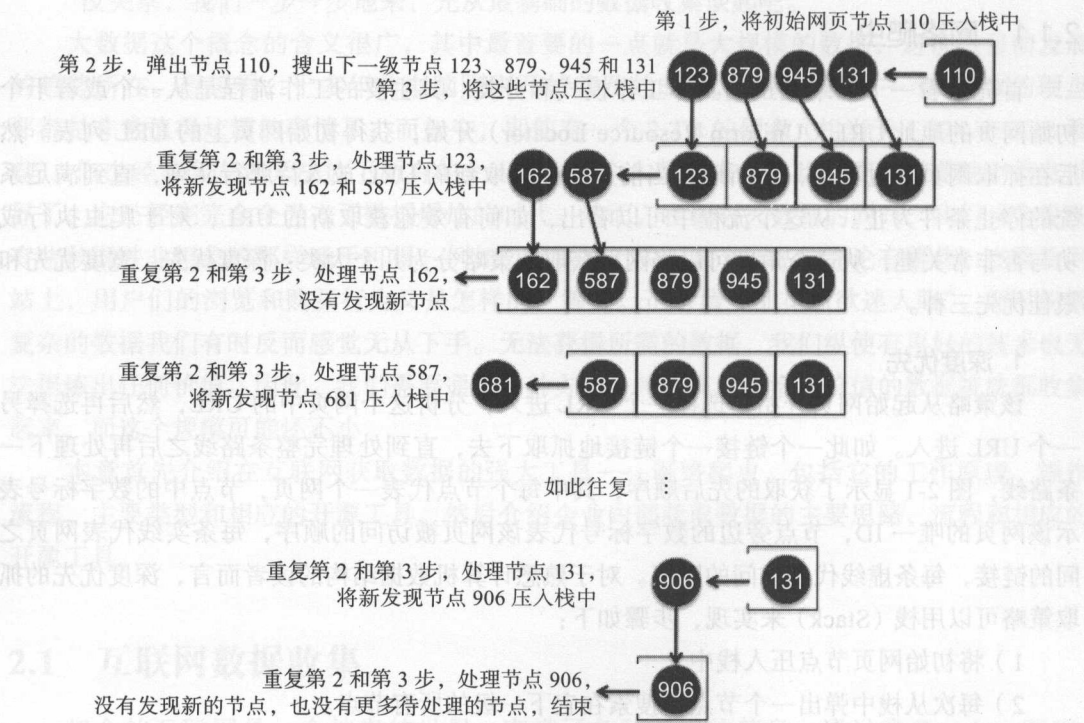


图 2-2 深度优先的栈实现

深度优先策略的设计较为简单，但是它面临一个最大的麻烦：陷入（Trapped）问题。门户网站提供的链接往往是最具价值的，评级也很高，但随着每一层的深入，网页的价值就会相应地有所下降。这就暗示了重要网页通常距离种子较近，而过度深入抓取到的网页却价值很低。同时，这种策略下抓取的深度将会直接影响抓取命中率及抓取效率，合理的控制深度是该种策略的关键。因此，相对于其他宽度优先和最佳优先策略而言，此种策略很少被使用。

## 2. 宽度优先

宽度优先又称为广度优先，这个策略是指在抓取过程中，只有在完成当前层次的搜索之后，才进行下一层次的搜索。该算法的设计和实现也是比较简单的，图 2-3 显示了获取的先后顺序。同样，图 2-3 中的每个节点代表一个网页，节点中的数字标号表示该网页的唯一 ID，节点旁边的数字标号代表该网页被访问的名次，每条实线代表网页之间的链接，每条虚线代表访问的顺序。相对于深度优先而言，宽度优先的策略更容易理解，就好比银行排队办理业务，先来的人先处理。因此，我们可以采用数据结构中的队列（Queue）来实现，步骤如下：

- 1) 将初始网页节点加入队列中。
- 2) 每次从队列首位取出一个节点，搜索在它下一级的所有节点。
- 3) 将第 2 步中新发现的节点加入队列的末尾。
- 4) 重复第 2 步和第 3 步，直到没有发现新的网页节点为止。

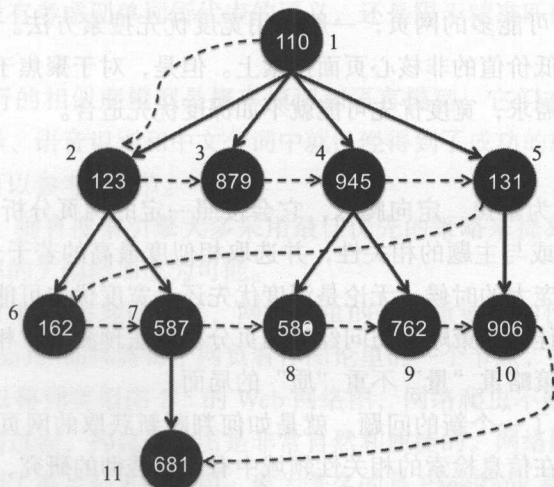


图 2-3 宽度优先的获取策略

队列的处理过程参见图 2-4。

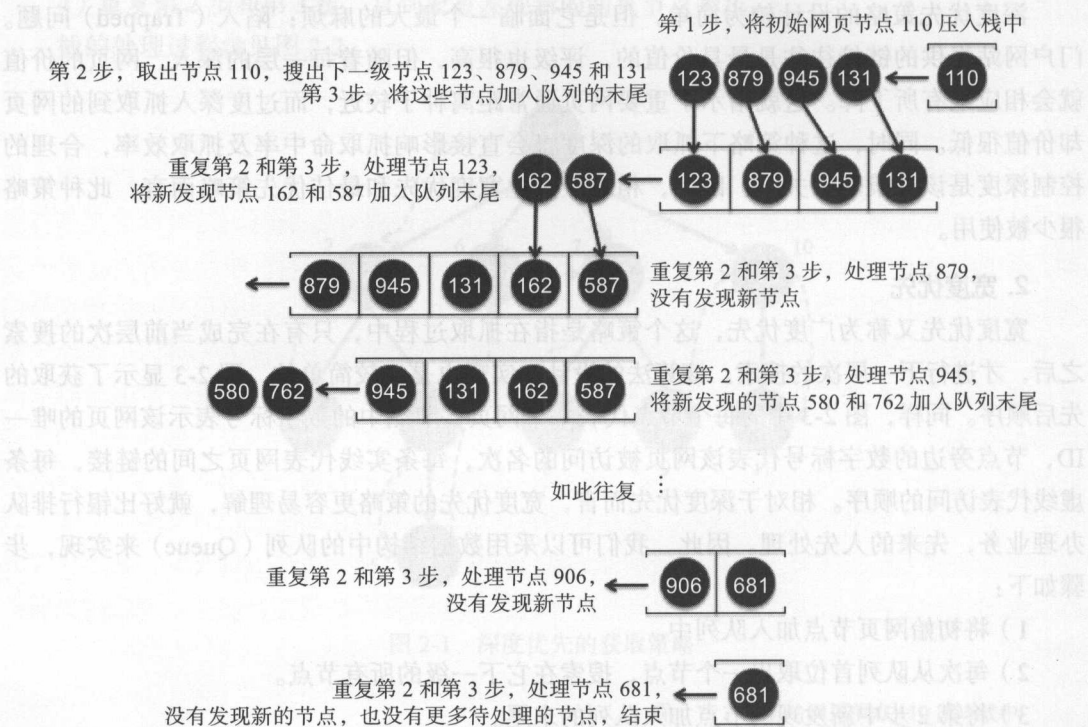


图 2-4 宽度优先的队列实现

当下，为了覆盖尽可能多的网页，一般使用宽度优先搜索方法。相对于深度优先的搜索，该策略不太会陷入低价值的非核心页面搜索上。但是，对于聚焦于某个网站，或者是网站局部频道的特殊爬取需求，宽度优先可能就不如深度优先适合。

3. 最佳优先

这种策略通常也称为聚焦、定向爬取，它会按照一定的网页分析算法，预测候选 URL 与目标网页的相似度，或与主题的相关性，并选取相似度最高的若干个 URL 进行抓取。当需要爬取的网页数量很庞大的时候，无论是深度优先还是宽度优先可能都不能高效地获取用户所关心的内容，而最佳优先策略只访问经过网页分析算法预测为“相关”的网页，在一定程度上缓解了之前两种策略重“量”不重“质”的局面。

不过，这里又引入了一个新的问题，就是如何判断新获取的网页和用户定义的候选网页之间的相似度呢？这在信息检索的相关性领域中有比较透彻的研究，本章先简单地介绍一下相似度计算的基础概念：相似度或相关性的衡量就是依据一定的模型，预测两个数据对象之间的相似程度，这里主要是注重文本的相似性。常见的模型有布尔模型、基于偏序的布尔模型、向量空间模型、概率模型和语言模型，等等。

布尔模型是最简单的方法，可用于比较不同的文本之间是否存在共同的关键词。如果



有就认为是相关或相似的,如果没有就认为不相关或不相似。不过,它的弱点就是对相关性的刻画不足。相关与否是一个模糊的概念,仅仅通过“相关”和“不相关”两个值来表示,未免过于绝对,况且也没有办法体现其中的区别。因此为了增强布尔模型,还需要考虑每个关键词的权重,如今使用最为普遍的是  $tf$ - $idf$  机制。这里的  $tf$  表示词频 (term frequency), 就是一个词  $t$  在文章  $d$  中 (或者是文章的某个字段中) 出现的次数。一般的假设是,某个词在文章中的  $tf$  越高,表示该词  $t$  对于该文档  $d$  而言越重要。同时,另外一个常用的是  $idf$ , 它表示逆文档频率 (inverse document frequency)。首先,  $df$  表示文档频率 (document frequency), 即文档集合  $c$  中出现某个词  $t$  的文档数量。也就是说,如果某个文档出现了词  $t$ , 那么  $df$  就增加并且只增加 1 次。一般的假设是,某个词  $t$  在文档集合  $c$  中的  $df$  越大,那么其重要性越低,反之则越高,因此将  $df$  转化为其倒数  $idf$  来表达这个理念。如此一来,布尔模型也能获得不同的相关性得分。

下面再介绍一个比排序布尔模型稍微复杂一点的向量空间模型。此模型的重点就是将某个文档转换为一个向量,向量的维度是单词,而每个维度的值通常也是用  $tf$ - $idf$  来表示的。这样,相关性问题就转化为计算查询向量和文档向量之间的相似度问题了。在实际处理中最常用的相似性度量方式是余弦距离。其好处是,其值正好是一个介于 0 到 1 之间的数,如果向量一致就是 1,如果正交就是 0,符合相似度百分比的特性。相对于标准的布尔数学模型,向量空间模型具有不少优点,包括基于线性代数的简单模型,非常容易理解;词组的权重可以不是二元的,例如可以采用  $tf$ - $idf$  这种机制;允许计算文档和索引之间的连续相似程度和基于此的排序等。当然,基本的向量空间模型也存在很多不足,例如对于很长的文档,相似度得分就会不理想;没有考虑到单词所代表的语义,还是限于精准匹配;没有考虑词在文档中出现的顺序,等等。

近几年另一些流行的相似度模型是概率模型和语言模型。它们本身其实并不是新兴的技术,之前在机器翻译、语音识别和中文分词中就已经得到了成功的应用。如果对这些模型的具体内容感兴趣,可以参考 5.2 节。

不管是哪种模型,垂直搜索引擎大多采用最佳优先的策略来提高定向爬虫的效率,使得数据收集和更新频率的大幅增加成为可能。

除了收集互联网中世界各地的信息,网络爬虫的一个重要附加价值就是可以帮助人们构建网络图 (Web Graph)。如果将每个网页看作图论里的一个节点,而网页间的超链接表示图论里的边,那么可以得到类似图 2-5 的 Web 网络图,网络爬虫不断获取新页面的过程就是遍历这些节点和边的过程,构建这种图是非常自然和便捷的。网络图的链接分析和挖掘在搜索引擎的排序中起到了举足轻重的作用,最为著名的是 PageRank 和 HITS 算法。

Google 的创始人拉里佩奇和谢尔盖布林于 1998 年发明了 PageRank 技术,并成为 Google 区别于当时搜索引擎的重要标志之一。其模型的假设是“随机冲浪者”,冲浪者从某张网页出发,根据 Web 图中的链接关系随机访问。在每个步骤中,他/她都会从当前网页的链出网页中随机选取一张作为下一步访问的目标。在整个 Web 图中,绝大部分的网页节



点都会有链入和链出。那么冲浪者就可以永不停歇地冲浪，持续在图中走下去。PageRank 的基本假设就是：在随机访问的过程中，越是被频繁访问的链接越重要。可以看出，每个节点的 PageRank 值取决于 Web 图的链接结构。假如一个页面节点有很多的链入链接，或者是链入的网页有较高的被访问率，那么它也将会有更高的被访问概率。同时，PageRank 引入了随机的跳转操作，也就是说假设冲浪者不按照 Web 图的拓扑结构走下去，只是随机挑选了一张网页进行跳转。这样的处理是类比人们打开一张新网页的行为，也是符合实际情况的，避免了信息孤岛的形成。

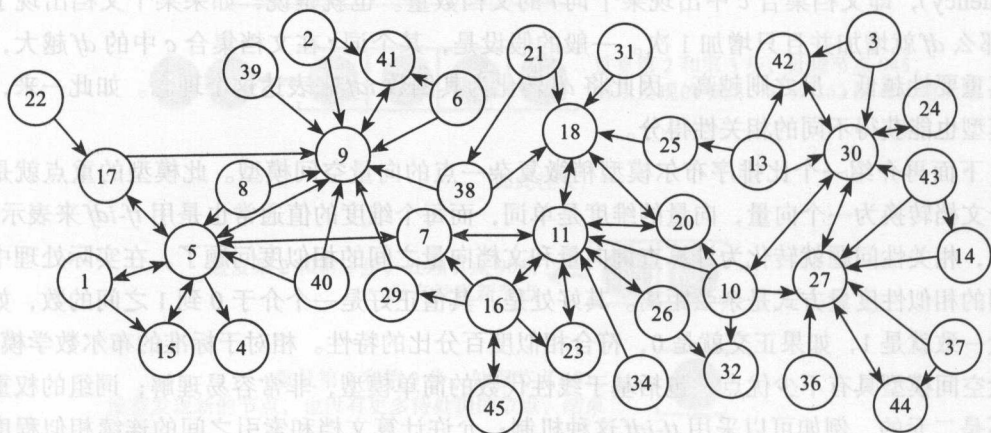


图 2-5 Web 形成的网络图样例

另一个有名的链接分析算法是 HITS，它是由康奈尔大学的 Jon Kleinberg 博士于 1998 年提出的。HITS 算法和 PageRank 最大的区别在于：它将每张网页节点的值分为两个，包括权威值（authority）和中心值（hub）。Web 上有很多人工编辑的导航性网页，如各种网址导航，其本身不会详细介绍某个主题，但是会指向富含权威内容的网站，这种称之为中心。相对而言，如果一张网页本身不具备导航功能，就是详细介绍某个主题的，那么就称之为权威。一张网页可以同时拥有中心和权威的身份，只是得分有高低。HITS 假设一个优质的中心网页会指向更多优质的权威网页，而一个优质的权威网页也会被更多中心网页所指向。因此，我们可以类似 PageRank 给出中心值和权威值的循环定义，并且通过迭代计算来求解，最终达到收敛的状态。关于链接的分析，更多具体的内容将会在 5.5 节中进一步探讨。

最后，我们来看一下爬虫实现的大致系统架构，如图 2-6 所示。

从图 2-6 可以看出，爬虫的工作流程为：首先根据种子网页的 URL，形成初始的待爬取 URL 集合，然后依次读取并从互联网上下载、保存、分析并获取该网页中新的 URL 链接。对于新的 URL，根据深度、宽度和最佳优先的不同策略，放入到待爬取的集合。如果是基于相似度的最佳定向策略，此时还需要考虑相似度的衡量。对于已经处理完毕的网页，将其内容存入数据库作为镜像缓存。而其 URL 地址则放入已爬取的集合，以避免重复。需

要注意的是,在很多应用场景下,网页爬取仅仅执行一次是不够的,需要定期更新其内容。因此已获取的集合需要定期更新,或者是针对不同类型的网页设置更为精细的更新计划。例如,像时事新闻这种网站,消息的及时性要求很高,因此可以加快对该类型站点的爬取频率,缩短更新的周期。随着整个流程不断的循环和往复,互联网上的数据会不断地收集和存储。由于互联网数据规模庞大的难以想象,所以爬虫通常需要不停地行走,以发现新的大陆,探索未知的信息世界。

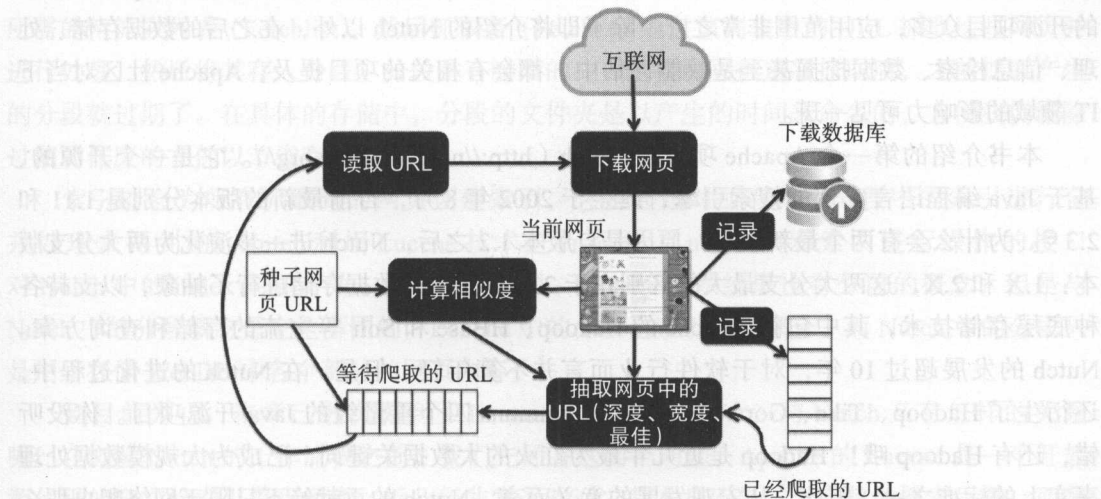


图 2-6 爬虫的基本架构和工作流程

“小明哥,那网络爬虫对于我们创业公司而言有什么价值呢?”

“对于创业公司而言,利用现有的外部资源是飞速发展的捷径。因此,网络爬虫将被广泛地运用在互联网世界的获取上。例如,对于你们的 O2O 电商创业而言,可以让爬虫获取某些地区的气候数据,以便于之后做基于天气预报的个性化推荐和精准营销。此外,爬虫还可以从主流电商网站爬取其商品目录结构,以及最新的市场价格,等等。”

在了解网络爬虫的基础知识和一些应用场景之后,我们可以尝试自己动手,搭建最简单的爬虫系统。当然,我们没有必要从零开始,下面即将介绍两个比较著名的开源系统——Apache Nutch 和 Heritrix,它们能帮助人们快速设计和实现自己的网络爬虫。

### 2.1.2 Apache Nutch 简介

“小明哥,什么是 Apache,什么是 Nutch?”

“嗯,我们先介绍 Apache 吧,后面的章节中会多处提及与其相关的开源项目。”

这里 Apache 是 Apache 软件基金会 (Apache Software Foundation, ASF), 及与其对应的开源社区、开源项目的统称。Apache 基金会是一个非营利性的组织,正式创建于 1999 年

7月，专门为开源软件项目的 Apache 团体提供运作支持。这个组织将自己定义为有着相同目标的开发者与用户团体，而不只是简单地共享在一个服务器上的一组项目。在它所支持的 Apache 项目与子项目中，所发行的软件产品都遵循 Apache 许可证。起初，这个组织的开发爱好者们聚集在一起，在美国伊利诺伊斯大学超级计算机应用程序国家中心开发的 NCSA HTTPd 服务器的基础上，开发与维护了一个名为 Apache 的 HTTP 服务器。由于需求的不断扩大，以 Apache HTTP 服务器为中心，Apache 社区启动了更多的并行项目，而这些项目也随着时间的推移、形势的变化而不断地启动、终止、拆分及合并。时至今日，Apache 旗下的开源项目众多，应用范围非常之广。除了即将介绍的 Nutch 以外，在之后的数据存储、处理、信息检索、数据挖掘甚至是效能评估中，都会有相关的项目提及，Apache 社区对当下 IT 领域的影响力可见一斑。

本书介绍的第一个 Apache 项目是 Nutch (<http://nutch.apache.org/>)。它是一个开源的、基于 Java 编程语言实现的搜索引擎，诞生于 2002 年 8 月，目前最新的版本分别是 1.11 和 2.3<sup>①</sup>。为什么会有两个最新版本？原因是自版本 1.2 之后，Nutch 进一步演化为两大分支版本：1.X 和 2.X，这两大分支最大的区别在于 2.X 对底层的数据存储进行了抽象，以支持各种底层存储技术，其中包括 Apache 的 Hadoop、HBase 和 Solr 等主流的存储和查询方案。Nutch 的发展超过 10 年，对于软件行业而言并不算年轻。但是，在 Nutch 的进化过程中，还衍生了 Hadoop、Tika、Gora 和 Crawler Commons 四个重量级的 Java 开源项目。你没听错，还有 Hadoop 哦！Hadoop 是近几年最为红火的大数据关键词，已成为大规模数据处理事实上的标准之一。因此，从宏观发展的意义而言，Nutch 的贡献绝不只限于网络爬虫那么简单。

此外，从功能的角度出发，Nutch 也不仅仅是一个爬虫系统，它还提供了运行搜索引擎所需的主要工具，包括全文索引和搜索。相对于那些商用的搜索引擎，Nutch 作为开放源代码的搜索引擎，系统架构更加透明，可定制化功能也更加强大，是入门学习的不错选择，其主要组成部分为爬虫（Crawler）、索引器（Indexer）和查询器（Searcher）。爬虫主要用于从网络上抓取网页并为这些网页建立索引，查询器主要是利用这些索引检索用户的查找关键词来产生查找结果，两者之间的接口是索引。如果想深入了解索引和查询的概念，可以先参考第 5 章的信息检索内容，里面有更为详尽的介绍和探讨。这里主要介绍爬虫相关的部分：涉及的数据文件，以及工作流程。

数据文件主要包括三类，分别是网络数据库（Web Database）、分段（Segment）和索引（Index），三者的物理文件存储在爬行结果目录下的 db 目录中，分别以 WebDB、segments 和 index 命名对应的子文件夹。Web Database，也称 WebDB，其所存储的是网页（Page）和链接（Link）实体。网页实体通过描述某个网页的特征信息来表征一个实际的网页。网页富含很多描述信息，为了提高效率，WebDB 中通过网页的链接 URL 和网页内容的 MD5（Message

① 本书中所涉及的软件最新版本编号均为作者写作该书时该软件对应的最新版本编号。



Digest Algorithm MD5)<sup>①</sup>两种方法对其进行标识。网页实体描述的网页特征主要包括网页内的链接数目、对此网页的重要度评分、抓取此网页的时间等相关信息。此外,链接实体描述的是两个网页实体之间的链接关系。WebDB 构成了一个所抓取网页的链接结构图,其中网页实体是图的结点,而链接实体则代表图的边。这就是我们上节所提到的网络图(Web Graph),对于链接分析非常关键。

爬虫的每次爬行都会产生很多个分段,每个分段内存储的是爬虫在单独一次抓取循环中抓到的网页。爬虫抓取时会按照一定的策略,从 WebDB 的链接关系中生成每次抓取循环所需的待抓取列表(Fetchlist),然后抓取者(Fetcher)通过该列表中的 URL 抓取这些网页并进行处理,然后将其存入分段。分段是有时限的,当这些网页被重新抓取后,之前抓取产生的分段就过期了。在具体的存储中,分段的文件夹是以产生的时间来命名的,方便我们删除过期的作废的文件以节省存储空间。

索引器会为所有抓取到的网页创建索引,它是通过对所有单个分段中的索引进行合并处理来得到的。Nutch 利用 Lucene 技术进行索引, Lucene 中对索引进行操作的接口对 Nutch 中的 index 同样有效。值得注意的是, Lucene 的系统里也有分段的概念,但是与 Nutch 中的分段概念是完全不同的。Lucene 中的分段是索引的一部分,而 Nutch 中的分段只是用于存储 WebDB 中各个部分的网页内容。

到目前为止,本章已数次提到了 Lucene,那么它是什么?与 Nutch 又有怎样的关系呢?在浏览 Nutch 的工作流程之前,我们先回答这两个问题。Apache 的 Lucene 是一个开放源代码的全文检索引擎程序库,近几年非常受欢迎。其本身不是一个完整的搜索引擎,而是为软件开发人员提供的一个简单易用的工具包,可用于在目标系统中实现全文检索的功能,或者是以此为基础建立起完整的全文搜索引擎。Nutch 在 Lucene 的基础之上做了进一步的扩展和封装,增加了爬虫的功能,同时还利用 Lucene 提供了文本索引和搜索的接口。因此,我们可以简单地理解为, Nutch 是一个可以直接运行的搜索引擎,不需要自己编写任何代码就能实现简单的全文搜索功能。不过,本章的重点是介绍网络爬虫,因此这里主要聚焦在 Nutch 获取数据的部分。对 Lucene 的使用细节有兴趣的读者可以参见第 5 章的相关内容。

在分析了爬虫工作中涉及的文件之后,接下来看看其抓取流程及这些文件在抓取过程中所扮演的角色。首先爬虫根据 WebDB 生成一个待抓取网页的 URL 集合 Fetchlist,接着抓取者线程 Fetcher 根据 Fetchlist 将网页抓取回来,如果下载线程有很多个,那么就生成很多个 Fetchlist,每个 Fetcher 对应一个 Fetchlist。然后爬虫用抓取回来的网页更新 WebDB,根据更新后的 WebDB 生成新的 Fetchlist,里面是尚未抓取的或新发现的网页链接,然后下一轮抓取循环开始。下面是这些子操作的功能描述及执行步骤,括号中是最基本的命令行。

1) 创建一个新的 WebDB (admin db -create)。

2) 将抓取起始链接写入 WebDB 中 (inject)。

① 计算机安全领域广泛使用的一种散列函数,确保信息传输的完整一致,常用于数据的完整性保护。

- 3) 根据 WebDB 生成 Fetchlist 并写入相应的分段 (generate)。
- 4) 根据 Fetchlist 中的链接抓取网页 (fetch)。
- 5) 根据抓取网页更新 WebDB (updated)。
- 6) 循环进行第 3 至第 5 步, 直至预先设定的抓取深度或宽度, 具体情况视获取策略而定。
- 7) 根据 WebDB 得到的网页评分和链接来更新分段 (updatesegs)。
- 8) 对所抓取的网页进行索引 (index)。
- 9) 在索引中丢弃有重复内容的网页和重复的链接 (dedup)。
- 10) 将 Lucene 索引分段中的数据进行合并, 生成用于检索的最终索引 (merge)。

至此, 我们了解了 Nutch 的工作模块和核心流程, 可以看出它是非常标准的爬虫实现。同时, 也要看到 Nutch 并不局限于爬虫, 对于索引和查询的相关内容, 我们可以放到第 5 章了解。

### 2.1.3 Heritrix 简介

Heritrix 也是一个开源的、使用 Java 编程语言开发的网络爬虫, 始于 2003 年年初, 最初用于对网上的资源进行归档, 建立网络数字图书馆。Heritrix 可以实现深度复制, 用于获取完整的、精确的站点内容, 甚至还包括图像及其他非文本内容。可以说, 它对任何内容都是“来者不拒”, 并且不会对页面内容进行修改。对于相同的 URL, 即使重新爬取也能不替换先前的内容。爬虫主要通过可视化的 Web 用户界面启动、监控和调整, 允许弹性地定义要获取的网页链接。其中, Heritrix 最出色之处在于其良好的可扩展性, 方便用户实现自己的抓取逻辑。目前可以在 SourceForge 开源站点下载其最新版本 (<http://sourceforge.net/projects/archive-crawler/>)。

从核心模块来看, Heritrix 主要有以下组件:

- 爬取范围: 通过规则, 配置当前应该在什么范围内抓取网页链接。例如, 选择 BroadScope 表示当前的爬取范围不受限制, 选择 HostScope 则表示爬取被限制在当前的同一台主机上, 不能爬取其他 IP 地址的主机。
- 边界开拓: 针对网页 URL 进行处理, 最主要的作用是决定下一个将被处理的 URL 是什么。这与深度优先、宽度优先这样的获取策略紧密相关。具体的操作包括跟踪哪些 URL 将被爬取, 哪些 URL 是已经被爬取过的, 选择下一个 URL, 剔除已经处理过的, 等等。
- 处理器链: 通过若干处理步骤获取 URL, 分析结果并将其传回给边界开拓组件。其中有 5 个子组件可以配置, 分别是预处理器、获取器、抽取器、写入器和后处理器。
  - 预取器: 做一些准备工作, 对抓取的先决条件做判断, 是整个处理器链的入口。例如, 对处理进行延迟和重新处理、否决随后的操作等。

- 获取器：主要是获得资源，解析网络传输协议。例如进行 DNS 转换等，并填写请求和响应的表单。
- 抽取器：解析获取器返回的内容，抽取感兴趣的部分，包括 HTML、CSS、JavaScript 等。解析完成后，新发现的网页链接也会被放入待处理列表，等待之后的继续爬取。
- 写入器：将爬取和抽取到的结果存储到磁盘，可以以压缩和非压缩镜像两种方式进行，压缩更省磁盘空间而非压缩则更省时间。
- 后处理器：做最后的维护和扫尾工作，如测试解析出的新链接中有哪些是不在爬取范围内的。

- Web 管理控制台：多数是单机的 Web 应用，通过内嵌 Java HTTP 服务器来实现展示。
- 爬虫命令处理：包含足够的信息，用于创建要爬取的 URL。
- 服务器缓存 (Servercache)：存放服务器的持久化信息，能够被爬行部件随时查阅，例如获取策略、抓取时间、IP 地址、历史记录等。

将 Heritrix 和 Nutch 做一个简单的对比，我们不难发现二者均为 Java 开源框架，它们的实现原理也基本一致：先遍历网站的资源，然后将这些资源抓取到本地。使用的方法都是分析网站中每一个有效的 URL 链接，并提交 HTTP 请求，从而获得相应的结果，生成本地文件及相应的日志信息等。当然，两者在具体实现和使用上还是有不同之处的：

- 从功能上看，Nutch 紧密集成了 Lucene，对于抓取内容的索引和检索是非常方便的。相比之下，使用 Heritrix 时，用户需要自己负责文件格式的转换、索引和检索等工作。不过，Heritrix 网络蜘蛛的功能更为强大，可以专注于网络信息的下载。
- 从对待爬取内容的处理方式上看，Nutch 只获取并保存可索引的内容，且可以对下载内容进行过滤修改，而 Heritrix 则适用于各种类型的信息，力求保持网页原貌。另外，Nutch 一般是通过刷新操作将旧内容替换为新内容的，而 Heritrix 则是不断追加新内容。
- 从用户使用来看，Nutch 采用命令行运行和控制。而 Heritrix 有 Web 控制管理界面，更为人性化一些。Nutch 的定制能力不算很强，而 Heritrix 可控制的参数比较多，只是配置起来有点麻烦。

## 2.2 内部数据收集

爬虫通过收集互联网每天不断产生新数据，能很好地帮助我们了解外面的世界。有时候，我们也需要关注公司和企业内部产生的数据，这些信息可能会有更高的商业价值。相对于互联网，内部数据的产生方式和时效性会有所不同，因此收集模式也会有所差异。首先，互联网上的数据分散在 Web 世界的各个角落，每时每刻都会有新的信息源诞生或消亡，没有一个全局的“地图”可供参考，因此只能依赖爬虫不断地行走和探索来将这个“地图”逐



步绘制，颇有探索新大陆的意味，这也是链接获取策略非常关键的原因。而对于内部数据，我们几乎无须考虑这个问题，无论是网站顾客的访问日志，还是公司业绩的销售数据，都会按照既有规划，存放在指定的位置，因此不会涉及链接的分析和获取策略。不过，从及时性的角度来看，内部数据的收集面临着更大的挑战。除了新闻、博客等特殊场景，网络爬虫的应用一般不会要求其进行快速处理，更适合于大规模的离线存储和事后分析。而内部数据极有可能关乎企业的整体运营和表现，是各种核心业务指标的分析来源，因此大多数情况下要求有一定的实时性保证。

也正是如上所述的主要差异，决定了内部数据收集的工作模式有所不同。首先我们来看收集的模型，内部数据的收集主要分为推送（Push）和拉取（Pull）两大类。两种模式都有各自的优缺点。如果选用推送模型，那么数据的实时性无疑是比较高的，不过数据收集的容量必须要大于高峰期数据产生的规模，否则如果生产源主动推送过来的数据不能得到及时处理，将会带来更多更复杂的后续问题，其弊端就会突显。如果选择拉取模型，主动权则掌握在收集端，收集的模块可以根据自己的节奏拉取数据，高峰期产生的海量数据会在一段时间内被慢慢地消化。如此一来，只要保证收集的能力高于数据产生的平均值就足够了。当然，这样的弊端是处理延迟明显增加了。图 2-7 和图 2-8 分别展示了这两种模式。在高峰期，推送的数据量可能过大，因此推送的模式中常常需要加上队列，进行异步处理，以达到缓冲的效果。图 2-7 也包含了这一模块。

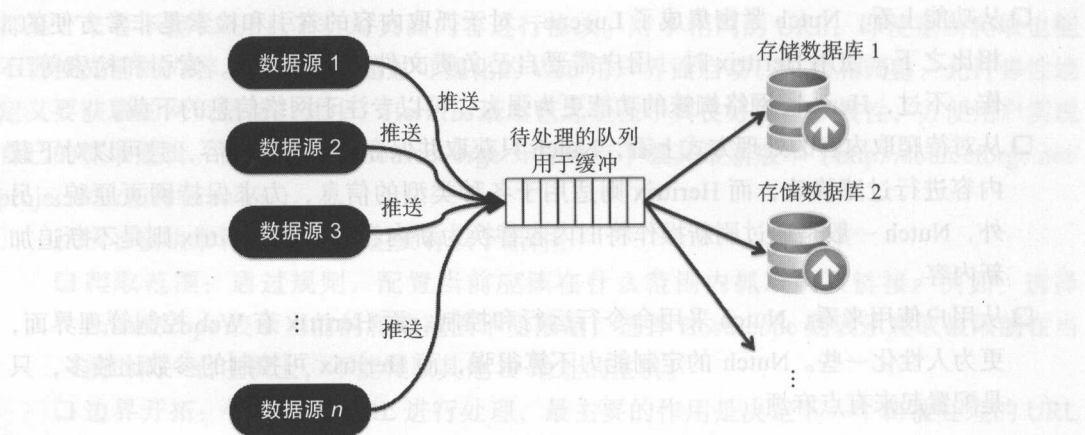


图 2-7 推送的收集方式

“大宝，你觉得这种站内数据的收集，可以怎样运用呢？”这次小明主动发问。

“啊哈，这个难不倒我。我们的创业一定是面向终端顾客的，那么他们在网站上的行为一定是数据分析的关键线索。所以，设计一套用户行为日志的跟踪和存储系统是非常重要的。这里的站内数据收集模块作为分布式日志的整合是再贴合不过的啦！”

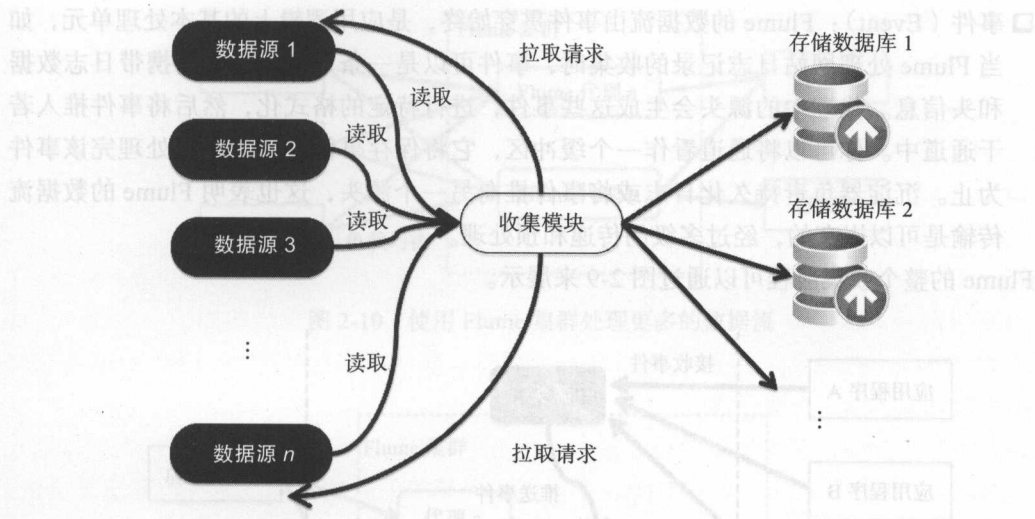


图 2-8 拉取的收集方式

## 2.2.1 Apache Flume 简介

Flume (<http://flume.apache.org/>) 是一个分布式、可靠和高可用的海量数据收集系统，目前最新的版本是 1.6.0。它同时采用推送和拉取这两种采集模式，其能力受到了业界的认可与广泛应用。它支持在系统中定制各类数据发送方，同时还支持对数据进行简单处理，然后写到各种可定制的数据接收方。Flume 最早属于知名的 Cloudera 公司，初始的发行版本被统称为 Flume-OG (Original Generation)，目前版本号是 Flume 0.9X 这种形式。随着 Flume 功能的扩展，Flume-OG 代码工程臃肿、核心模块设计不合理、核心配置不标准等缺点纷纷暴露出来，尤其是在 Flume-OG 的最后一个发行版本 0.94.0 中，日志传输不稳定的问题尤为突出。为了解决这些问题，2011 年 Cloudera 完成了 Flume 中里程碑式的改动，核心模块、核心配置及代码架构都得到了重构，改善后的新版本统称为 Flume-NG (Next Generation)，版本号都是 Flume 1.X 形式。与此同时，Flume 也被纳入 Apache 社区，Cloudera Flume 正式改名为 Apache Flume。本书后面提到的 Flume 如无特殊说明均指 Flume-NG。

Flume 的核心模块有三个：

- 源头 (Source)：负责接收数据的模块，它定义了数据的源头，从源头收集数据，传递给通道。源头还可用于接收其他 Flume 代理的沉淀器传输过来的数据。
- 沉淀器 (Sink)：批量地从通道读取并移除数据，并将所读取的内容存储到指定的位置。
- 通道 (Channel)：作为一个管道或队列，连接源头和沉淀器。

通过这几个模块，形成了如下重要概念：

- 代理 (Agent)：Flume 运行在服务器上的程序，是最小的运行单位。每台机器只会运行一个代理，其中可包含多个源头和沉淀器。



❑ 事件（Event）：Flume 的数据流由事件贯穿始终，是应用逻辑上的基本处理单元，如当 Flume 处理网站日志记录的收集时，事件可以是一条日志记录，它携带日志数据和头信息。代理中的源头会生成这些事件，进行特定的格式化，然后将事件推入若干通道中。你可以将通道看作一个缓冲区，它将保存事件直到沉淀器处理完该事件为止。沉淀器负责持久化日志或将事件推向另一个源头，这也表明 Flume 的数据流传输是可以嵌套的，经过多级的传递和预处理。

Flume 的整个收集流程可以通过图 2-9 来展示。

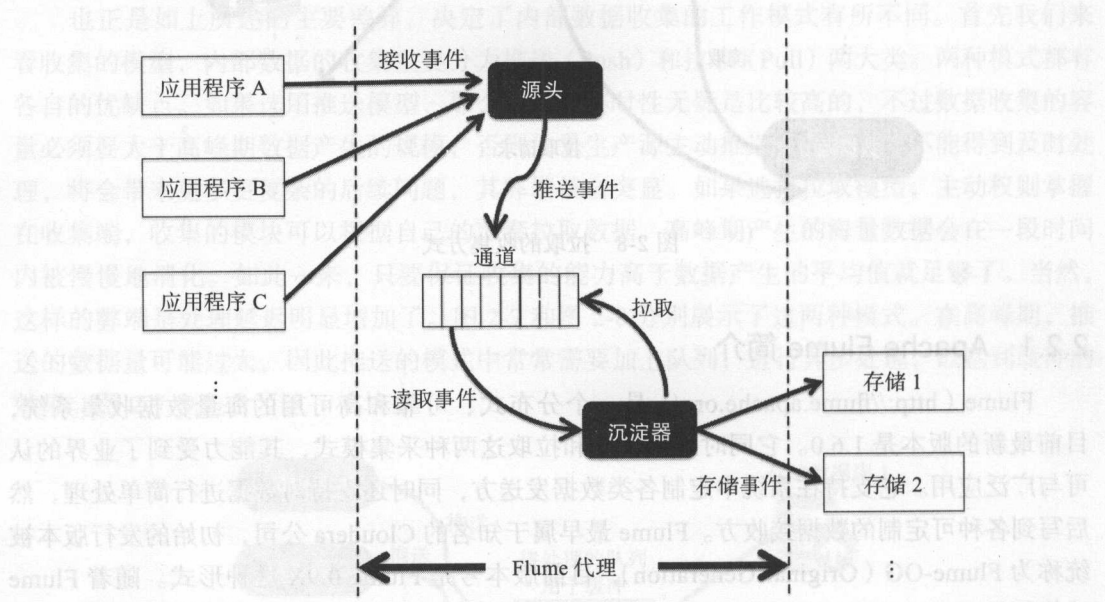


图 2-9 Flume 工作的基本流程

在图 2-9 中，我们可以将源头想象成为一个水龙头，沉淀器是一个水桶，而通道就是水管。水管两头分别接上水龙头和水桶，当水龙头打开，水就源源不断地通过水管流入水桶。

Flume 的一大优势在于它是集群化管理，如果需要采集的应用过多的时候，单个 Flume 的代理可能就无法处理了，这时就需要更多的代理来组成集群，图 2-10 显示了大体的架构，其中从应用程序端到集群就需要做流量的负载均衡。类似的比方，这里可以想象成水龙头有太多的水需要放出，一根水管远远不够，因此需要接多根水管用于传送。

前文也提到过，Flume 的数据流是可以通过多级嵌套来进行传输的，图 2-11 就体现了这样的架构。如此架构的优势在于，可以将不同的处理逻辑分层，以便于开发、测试和管理。同时，也能更好地控制数据流缓冲的节奏。同样，我们可以认为是通过管道连接器将多段水管连接起来了。每个连接器还可以加入不同的净水处理模块，比如，让第 1 段和第 2 段水管连接器进行活性炭吸附杂质，而第 2 段和第 3 段的水管连接器进行紫外线杀菌，等等。

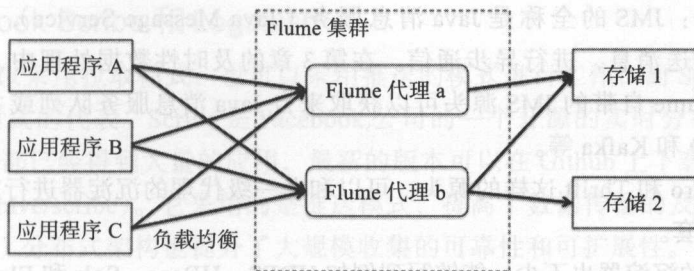


图 2-10 使用 Flume 集群处理更多的数据流

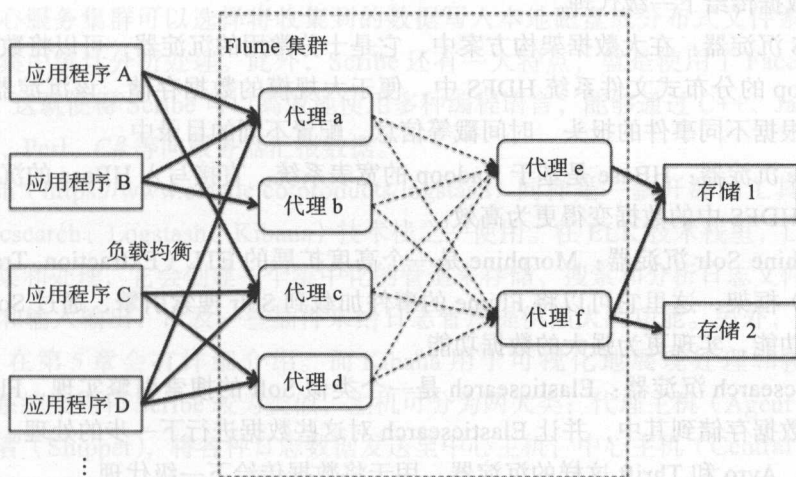


图 2-11 层次型的 Flume 集群架构

从上述的模块和流程中可以看出，源头、沉淀器和通道的实现至关重要。好消息是，对于三大模块，Flume 已经为我们实现了很多的基本功能，下面来快速浏览一下。

Flume 的源头主要包括：

- ❑ **Spooling Directory 源头：**在一些场景中，应用数据产生后会被存入本地的文件中，文件中的一行或若干行组成一个逻辑单元。但是不同的应用会导致不同的字段定义和格式，而你又无法修改这些应用本身，如何将这些信息整合是一个令人头疼的问题。这个时候 Spooling Directory 源头就有了用武之地，它是非常简单和常用的源头，会监视指定目录的变化，从这些目录的文件中读取所需要的数据，并且进行必要的预处理，将不同数据源的内容转化为对应于 Flume 的事件。由于 Spooling Directory 源头是磁盘 I/O 读取密集型的，所以它的性能通常不会很高。
- ❑ **HTTP 源头：**该源头可以通过 HTTP 的 POST 方式接收数据。从客户端的角度来看，它的表现就像 Web 服务器一样，同时还接收 Flume 的事件。

□ JMS 源头：JMS 的全称是 Java 消息服务 (Java Message Service)，用于在分布式系统中发送消息，进行异步通信。在第 3 章的及时性数据处理中，我们还会详细介绍。Flume 自带的 JMS 源头可以获取来自 Java 消息服务队列或主题的数据，如 ActiveMQ 和 Kafka 等。

□ 嵌套：Avro 和 Thrift 这样的源头，可以和上一级代理的沉淀器进行对接，实现代理的多级嵌套。

Flume 封装的沉淀器也不少，能够写到例如 HDFS、HBase、Solr 和 Elasticsearch<sup>①</sup>等存储和检索引擎中。由于它们在 Flume 流程中通常都是最终点，因此这些类型一般被称为终端沉淀器。另外一些类型，如 Avro 和 Thrift 沉淀器，将与之前介绍的 Avro 和 Thrift 源头对接，用于将数据传给下一级代理。

□ HDFS 沉淀器：在大数据架构方案中，它是十分常用的沉淀器，可以将数据直接写入 Hadoop 的分布式文件系统 HDFS 中，便于大规模的数据存储。该沉淀器非常灵活，可以根据不同事件的报头、时间戳等信息，配置不同的目录中。

□ HBase 沉淀器：HBase 是基于 Hadoop 的宽表系统。直接写入 HBase 的沉淀器将使得查询 HDFS 中的数据变得更为高效。

□ Morphine Solr 沉淀器：Morphine 是一个高度扩展的 ETL (Extraction, Transform and Load) 框架，这里它可以将 Flume 的事件加载到 Solr 搜索引擎，通过 Solr 的索引和查询功能，实现更为强大的数据功能。

□ Elasticsearch 沉淀器：Elasticsearch 是一个类似 Solr 的搜索引擎实现，Flume 同样可以将数据存储到其中，并让 Elasticsearch 对这些数据进行下一步的处理。

□ 嵌套：Avro 和 Thrift 这样的沉淀器，用于将数据传给下一级代理。

Flume 自带两种通道：

□ 内存通道：该通道是内存中的队列，源头从它的尾部写入数据，而沉淀器从它的头部读取数据。由于都是内存操作，因此内存通道可以支持非常高的吞吐量。不过由于是非持久化<sup>②</sup>的方式，存在丢失数据的风险。

□ 文件通道：该通道会将事件都写入到磁盘中，以持久化的方式保存。这样数据就不会因为突然宕机或断电而丢失，而且海量数据的存储成本也较低，不过其性能远远不及内存通道。综合来看，如果对于数据的丢失无法容忍，并且不在意数据处理的速度，那么文件通道是理想的选择。相反，如果对系统反应速度要求很高，而且能允许一定程度的数据丢失，那么就可以选择内存通道。

还有一些源头和沉淀器通道的实现在这里没有被提及。更为重要的是，我们不要忘记 Flume 是开放源代码的，这就意味着开发者们完全可以自定义这些模块的功能和实现。

① 如果你对这些名词陌生，不用担心，后面相应的章节会有介绍，此处可以暂时忽略。

② 数据的持久化和非持久化概念，会在第 3 章的开头进行介绍。



## 2.2.2 Facebook Scribe 和 Logstash

Flume 既可以采用拉取模式,也可以采用推送的模式进行工作,而 Scribe 和 Logstash 则是采用推送模式的代表。Scribe 是 Facebook 公司的一个开源的实时分布式数据收集系统,在其公司内部已经得到大量的应用,最新的版本可以在 Github 上下载 (<https://github.com/facebookarchive/scribe>)。它采用的是推送模式,提高了数据传输的及时性。另外,C/S (Client-Server) 分布式架构也提升了大规模收集的可靠性和可扩展性。由于采用的是推送模式,因此使用者可以在不同的服务器节点上安装 Scribe 服务,从各种数据源上收集数据,之后将它们放到一个共享队列上,并推送到后端的中心存储系统上。当中心服务不可得到时,本地的 Scribe 服务会把收集到的数据暂时存储到本地,等中心服务恢复以后再进行上传。中心服务集群可以选择将收集到的数据写入本地磁盘或分布式文件系统上,以便于日后进行集中统计分析处理。此外,Scribe 还有一大特点,就是使用了 Facebook 公司的 Thrift 框架,这就使得 Scribe 可以高效地使用多种编程语言,能够通过 C++、Java、Python、PHP、Ruby、Perl、C# 等向服务器汇报数据。

Logstash (<https://www.elastic.co/products/logstash>) 同样是一款开源的工具,经常作为 ELK (Elasticsearch、Logstash、Kibana) 技术栈之一使用。在 ELK 技术栈里,Logstash 常用于日志的收集和处理,它会创建一个集中化的管道来存储、搜索和分析日志文件。它使用内建的过滤器和输入输出,以及一些插件来给日志管理提供强大的功能。另外,Elasticsearch 用于搜索,在第5章会有详细介绍。而 Kibana 用于可视化地展现处理和搜索的结果。Logstash 的系统架构和 Scribe 较为类似,主机可分为两大类:代理主机 (Agent Host),作为事件的发送者 (Shipper),将各种日志数据发送至中心主机;中心主机 (Central Host),可运行包括中间转发器 (Broker)、索引器 (Indexer)、搜索和存储器 (Search and Storage)、Web 界面端 (Web Interface) 在内的各个模块,以实现日志数据的接收、处理和存储。Logstash 的主要优势在于日志存储的方式更加灵活,对日志数据有更好的语法分析功能,易于安装、可扩展、性能良好等。

## 2.3 本章心得

“看来仅仅是数据的采集,就有不少学问啊。”

“是的,对于互联网上的公开数据,我们可以通过爬虫这个强大的工具来获取所需的内容。深度获取、宽度获取和最佳 (聚焦或定向) 优先是不同的爬取策略,让爬虫可以不断地发现‘数据的新大陆’。在数据的爬取方面,Nutch 和 Heritrix 都是不错的开源系统,可以帮助我们快速实现爬取。对于企业内部的数据而言,数据源的发现不再是问题,更要强调的是数据收集和传输的及时性。Flume、Scribe 和 Logstash 等开源方案,提供了拉取或推送的采集模式,提供了不同的特性和选择。”



“小明哥，感谢你今天关于数据采集的介绍，这下我们不愁没有大量的数据玩了。对了，还有一个问题，就是这么多的数据如何才能存放下来呢？”

“存放确实很关键。无论是互联网，还是企业内部收集到的数据，都可以使用像 Hadoop 这样 NoSQL 的大数据解决方案来存储。别急，我们会在下一章来探讨这个问题。如果你对数据收集本身还想做进一步了解，可以参考下面的图书。”

## 2.4 参考资料

### □《开发自己的搜索引擎——Lucene+Heritrix（第2版）》

作者：邱哲，符滔滔，王学松

本书对介绍了如何使用 Heritrix 构建自己的爬虫系统，同时对于 Lucene 的索引和查询进行了阐述。

### □《解密搜索引擎技术实战：Lucene & Java 精华版（第2版）》

作者：罗刚等

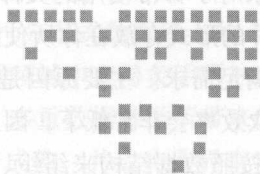
本书更深入一步探讨了如何通过编程来打造一个爬虫系统，比较适合程序员老手，也包含了 Lucene 和 Solr 系统的介绍。

### □《Flume：构建高可用、可扩展的海量日志采集系统》

作者：Hari Shreedharan

译者：马延辉，史东杰

如果需要深入了解 Flume 的细节，那么这本书会是一个不错的选择。作者是 Flume 的开发者之一，本书对 Flume 架构、配置和部署都有详细的阐述。



## 第3章

## Chapter 3

## 数据存储

第2章描述了如何获取企业内外的数据。当时，大宝提出了一个很好的问题：如今的互联网和公司内部每时每刻都在产生大量的信息，面对与日俱增的海量数据，我们应该如何存放它们呢？本章将带领大家一探究竟。

在此之前，先要解释一下第2章中提到的“持久化”和“非持久化”存储的概念。大家平时可能都有过这样的经历：老板交代下班前一定要完成一个重要的文档。你欢快地写了半天，突然电脑死机了，电脑重启后，再次打开文档却发现最近半小时的内容竟然没来得及保存，顿时石化……在这个案例中，半小时前你点击了“保存”按钮，所有的修改内容都保存到了磁盘（或者叫硬盘）上，这个用计算机专业的术语来说就叫“持久化”。持久化之后，就表示这部分内容真真切切地写入磁盘了，哪怕系统死机或断电，都不会影响这部分的数据。只要磁盘硬件本身没有出现问题，就能读取。相对的，最后那半小时你所做的修改都放在内存中，系统重启或断电后，内存的数据是无法保留的，一定会被清除，这就是“非持久化”。非持久化虽然不能像持久化那样永久地存放数据，但它的一大优势在于读取和写入的速度非常惊人。很难想象，如果没有内存我们的计算机系统会慢到何种地步。所以，在日常应用中，如果需要存储大量的资料，必须选择容量大的硬盘。如果要提升计算机的运行速度，提升硬盘的容量并没有帮助，这时应该提升内存的容量。当然，如果经费允许，两者都大一些，也没什么不好。

大数据存储和个人计算机应用有着类似的道理。在本章余下的部分中，将会从持久化和非持久化两个大的方向，分别介绍一些时下主流的大数据存储系统和解决方案。

### 3.1 持久化存储

对于持久化的存储而言，最关键的概念就是文件系统和数据库系统。对于个人计算机

而言，相信大家对于日常使用的文件系统已经非常熟悉了。任何文档、图片、影视、歌曲、游戏都是以文件的形式存放在你所使用的操作系统中的。可是，对于工程开发而言，文件系统还远远不能满足需求。主要原因是大量的文件没有很好地组织，也缺乏对象之间的关联。这样的数据读取效率会非常低下，因此数据库（Database）应运而生。数据库诞生于 20 世纪 70 年代，它是按照数据结构来组织、存储和管理数据的仓库。其中，最为常用的是经典的关系型数据库，它是建立在关系模型的基础上的，借助于集合代数等数学概念和方法来处理数据。现实世界中的各种实体及实体之间的各种联系均可以用关系模型来表示，其中的核心元素之一是 ER 图（Entity Relationship Diagram），它对于我们理解关系型数据库很有帮助。在 ER 图中，具体事物被划分为各个实体（Entity），每个实体通过多个属性（Attribute）进行描述，不同的实体之间又通过关系（Relation）进行连接。假设我们要描述一个公司的员工参加公司俱乐部的案例，可以用如图 3-1 所示的 ER 图来表示。

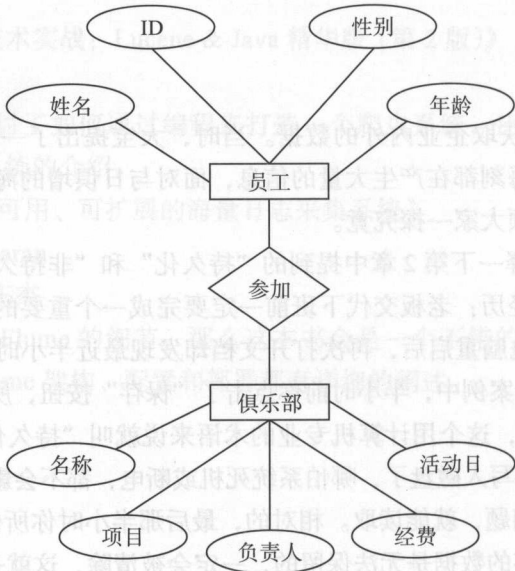


图 3-1 表示员工参加公司俱乐部的 ER 图

其中，员工有 4 个属性：姓名、员工 ID、性别和年龄，而俱乐部有 5 个属性：名称、项目、负责人、经费和活动日。它们之间是员工“参加”俱乐部的关系。可以看出，关系型模型中的数据都有良好的结构，因此关系型数据往往也称为结构化的数据。对于关系型数据的检索和操作，常常还需要使用 SQL（Structured Query Language，结构化查询语言）。它是专为数据库建立的操作命令集，是一种功能齐全的数据库语言。在使用 SQL 时，使用者只需要发出“做什么”的命令，而无须考虑“怎么做”的具体逻辑。SQL 功能强大、简单易学、使用方便，已经成为数据库操作的基础，目前几乎所有的数据库均支持 SQL。

可是，无论是文件系统还是数据库系统，在大数据时代都面临着巨大的挑战。单台计

计算机已经无法满足数据存储和处理的需求，必须要采用集群化的方案。那么，如何高效地读取这些分布在多台机器上的文件？又是如何高效地执行分布式数据库上的 SQL 查询的呢？最近几年，NoSQL（Not Only SQL）这个名字在大数据领域中变得炙手可热，它泛指非关系型的数据库，其产生就是为了帮助解决海量数据集合、多重数据种类所带来的大数据应用难题。随着 NoSQL 的流行，有不少具体的方案开始逐步成熟，并成为业界事实上的标准。这里将介绍知名的分布式文件系统 HDFS（Hadoop Distributed File System）和对应的分布式非关系型（非结构化）数据库系统 HBase，以及另一个非关系型的数据库 MongoDB。

### 3.1.1 Hadoop 和 HDFS

在第2章介绍 Apache Nutch 的时候，就曾提到过 Hadoop 起源于 Nutch。2003 年，Google 发表了一篇论文描述他们的分布式文件系统 GFS（Google File System），为 Nutch 攻克数十亿网页的存储难题提供了方向。Nutch 和 Lucene 的创始人 Doug Cutting 受到启发，和团队一起开发了 Nutch 的分布式文件系统 NDFS（Nutch Distributed File System）。2004 年，Google 又发表了一篇重量级的论文《MapReduce：在大规模集群上的简化数据处理》（“MapReduce: Simplified Data Processing on Large Clusters”）。之后，Doug Cutting 等人开始尝试实现论文所阐述的计算框架 MapReduce。此外，为了更好地支持该框架，他们还将其与 NDFS 相结合。2006 年，该项目从 Nutch 搜索引擎中独立出来，成为如今的 Hadoop（<http://hadoop.apache.org>）。两年之后，Hadoop 已经发展成为 Apache 基金会的顶级项目，并应用到很多著名的互联网公司中，目前其最新的版本是 2.x<sup>①</sup>。

如今的 Hadoop 系统已经可以让使用者轻松地架构分布式存储平台了，开发和运行大规模的数据处理应用，其主要优势如下。

- 透明性：使用者可以在不了解 Hadoop 分布式底层细节的情况下，开发分布式程序，充分利用集群的威力进行高速运算和存储。
- 高扩展性：扩展分为纵向扩展和横向扩展，纵向扩展将增加单机的资源，总会达到瓶颈；而横向将增加集群中的机器数量，获得近似线性增加的性能，不容易达到瓶颈。Hadoop 集群中的节点资源，采用的就是横向方式，可以方便地进行扩充，并获得显著的性能提升。
- 高效性：由于采用了多个资源并行处理，使得 Hadoop 不再受限于单机操作（特别是较慢的磁盘 I/O 读写），可以快速地完成大规模的任务。加上其所具有的可扩展性，随着硬件资源的增加，性能将会得到进一步的提升。
- 高容错和高可靠性：Hadoop 中的数据都有多处备份，如果数据发生丢失或损坏，能够自动从其他副本（Replication）进行复原。同理，失败的计算任务也可以分配到新的资源节点，进行自动重试。

① 由于历史的原因，Hadoop 的版本号有点复杂，同时存在 0.x、1.x 和 2.x，具体可以参见 Apache 的官网。



□ 低成本：正是因为 Hadoop 有良好的扩展性和容错性，所以没有必要再为其添置昂贵的高端服务器。廉价的硬件，甚至是个人计算机都可以成为资源节点。

Hadoop 的发展历史决定了 Hadoop 框架的最核心元素就是 HDFS 和 MapReduce。HDFS 为海量的数据提供了存储，而 MapReduce 为海量的数据提供了计算。本章将侧重于 HDFS 的介绍，更多关于 MapReduce 的介绍将放在第 4 章，该章节专门讨论数据的处理。

HDFS 是受到 Google 的大型分布式文件系统 GFS 的启发而开发出来的。Google 的数据中心采用了廉价的 Linux PC 服务器构建集群，而 GFS 正是这个数据中心的存储系统，它隐藏了底层的负载均衡、切片备份等实现细节，让复杂性透明化，并提供了统一的文件系统访问接口。由于一脉相承，HDFS 也有类似于 GFS 的特性和架构，适合部署在低廉的硬件上，容错性也比较高。同时，它还能提供高吞吐量的读取，适合那些有着超大数据集的应用场景。

为了更好地理解 HDFS 的体系架构，让我们先思考一个虚拟的案例：假设有一个名叫 Hadoop 的快消品公司，专门采用 Flume 公司提供的水源来生产各式饮用水和饮料，每天都有大量的纯净水通过 Flume 公司的车队运送到 Hadoop 公司的仓库。起初，Hadoop 公司的规模并不大，一个仓库就足以满足业务的生产需求。随着市场的不断扩张，业绩蒸蒸日上，仓库已经无法放下更多的原材料了。但这个仓库建在市区，周边没有多余的空地供仓库扩建，如果再购买市区现有的其他仓库，成本又太高。怎么办呢？公司的高层想到了在市区周围的几个郊县再多建几处仓库。10 多个仓库盖好了，生产的原材料有了足够的场地可以存放。可是这个时候 Flume 公司傻眼了，我们的纯净水到底要发往哪个仓库呢？如果不确定清楚，就有可能出现这样的情况：送到 1 号仓库，发现 1 号已经满仓，无法收货，而 2 号、3 号和 4 号仓库等却都是空的。为了便于协调，Hadoop 公司又成立了专门的仓库管理部门，负责收集各个仓库的负载情况，并及时与 Flume 公司的运输队进行沟通，整个流程渐渐变得顺畅。可是好景不长，高层很快发现了新的问题。原来，生产不同的饮料需要不同类型的水源，之前的做法是将某类水固定地存放在指定的仓库，例如 A 类水永远存放在 3 号仓。但是 3 号仓库总是发生意外状况，不是被小偷盗窃，就是大型储水桶发生渗漏，结果导致某些饮料的生产常常被耽误。该怎么办呢？类似的情况，将来会不会也发生在其他仓库呢？高层领导伤透了脑筋。关键的时刻，有人献上了妙计：要改变这种固定存放的模式，A 类水分批放在 3 号、6 号和 8 号库。这样，即使 3 号库的 A 类纯净水缺货，还可以从 6 号和 8 号库及时调拨，不至于影响饮料的生产。然后，再从 Flume 公司购买新的 A 类水放入 3 号库。这个策略果然生效，停产的风险大大降低，之后其他仓库发生类似的问题时也通过这个方案得到了很好的解决。

这个案例很容易理解，从中可以总结出如下 3 种行为：扩建仓库、增加协同部门和多仓备货。其实，HDFS 架构和仓库的原理非常类似，下面看看这三种行为，分别对应于 HDFS 的哪些概念。

□ 扩建仓库：单个仓库的扩建，就是所谓的纵向扩展。纵向扩展很容易达到瓶颈，建立仓库是这样，计算机系统也同样如此。例如单机的硬盘和内存，不可能无限制地被加大。这时就需要考虑横向扩展了，在郊区新建一个仓库，在计算机系统中就是

增加新的机器作为资源节点，不过这些机器节点存储的不再是纯净水，而是数据。在 HDFS 中，这些存储数据的节点被称为数据节点（Data Node）。

□ 增加协调部门：协同部门可以实时收集各个仓库的运作情况，并决策将进货存放在哪里更为合适。在 HDFS 中，扮演这个角色的节点称为命名节点（Name Node），它维护着系统中的大量元数据，负责管理文件系统的命名空间（Name Space）和控制外部的访问，包括打开、关闭、重命名文件或目录，将数据块映射到具体的数据节点等。随着协同部门职能重要性的日益增加，总公司可能还会将其管理内容进行备份，这就是次要命名节点（Secondary Name Node）。次要命名节点和命名节点的区别在于，它不会与数据节点和其他任务节点沟通，也不接收 HDFS 上的任何变化记录。次要命名节点最主要的目标就是与命名节点通信，根据配置定期地获取命名节点上的 HDFS 元数据快照，因此效率是非常高的。

□ 在多个仓库中进行备货：意外总是会发生，为了防止意外导致缺货的情况发生，可以采用的一项策略是将货物存储在地多的仓库中。在 HDFS 中也有同样的理念，这就是备份或副本（Replication）。存储在数据节点上的数据库可以有多个副本，并分发到其他节点上。这样在某个数据节点上丢失的数据，可以在其他数据节点上找到并恢复。容错性得以提升。

通过这个案例的比喻，也可以很容易地理解 HDFS 分布式文件系统一个重要的运用场景，就是与第 2 章中介绍的数据收集相互集成，保存互联网和大型企业内部每天产生的海量数据内容。理解了这些基本的概念，我们就可以画出图 3-2 来展示 HDFS 的工作原理了。

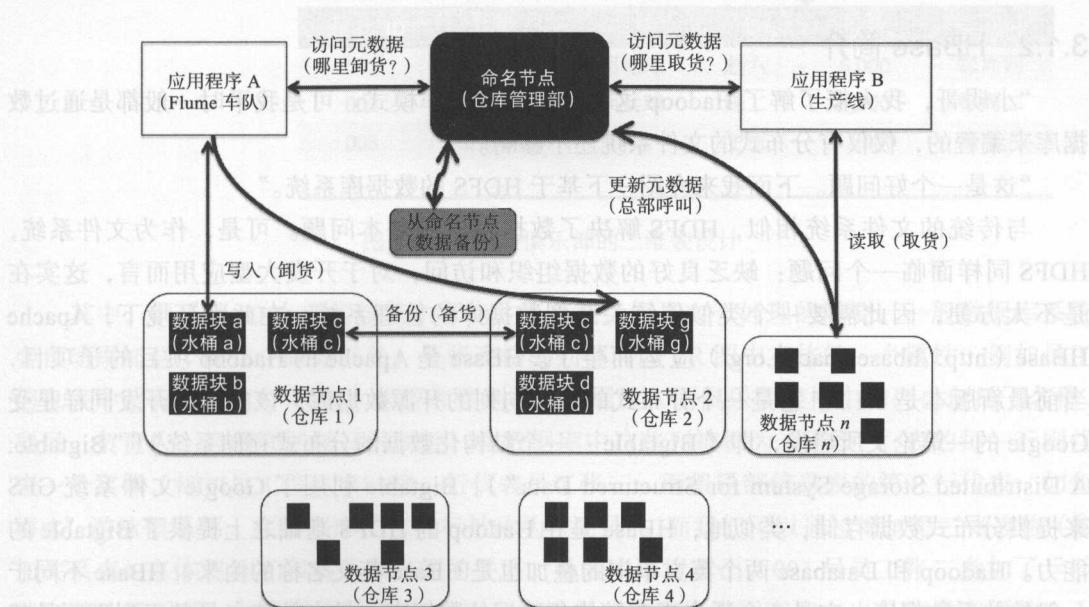


图 3-2 类似仓储系统的 HDFS

不过，在了解了 HDFS 的大致工作后，我们也会发现，HDFS 存在如下几个弱点：

❑ 不适合实时性很强的数据访问。试想一下，还是将不同类型的水源分散在不同的仓库存储，而现在有一款神秘的饮料，其制作工艺极其复杂，需要多种类型的净水作为原材料。因此，必须要到多个仓库取货，最后才能组装生产，那么 Hadoop 公司是不是很累？就是说，对于一个应用的查询，其对应的数据通常是分散在 HDFS 系统中的不同数据节点上的。为了获取全部的数据，需要访问多个节点，并且将不同部分的结果在网络中传输，最后进行合并。可是，网络传输的速度，相对于本机的硬盘和内存读取，都要慢很多，因此就拖累了数据查询的执行过程。

❑ 无法高效存储大量小文件。如果 Flume 公司有一天突发奇想，想恶搞一下 Hadoop 公司，送水的时候不再用超级大桶，而是用小奶瓶装，那么 Hadoop 公司的仓库管理部门绝对要跳脚。本来是 1 万桶水，结果顷刻之间变成了 1 亿个小奶瓶，还需要跟踪它们都存储在哪些仓库，简直是要将人逼疯。对于 HDFS 而言，命名节点承担的就是协调和管理工作，如果存在太多的琐碎文件，就意味着有庞大的元数据需要处理，这无疑大大增加了命名节点的负载。命名节点检索的效率会明显下降，最终也会导致整体的处理速度放缓。

此外，HDFS 对多用户的写入及文件任意修改的支持也不足。文件并发时的写入者只有 1 个，而且写操作只能在文件末尾追加新的数据，还不能在文件的任意位置进行插入。但不管怎样，HDFS 整体上还是拥有良好的分布式文件系统设计，对 Hadoop 及其生态体系的流行起到了关键的作用。

### 3.1.2 HBase 简介

“小明哥，我大概了解了 Hadoop 这种分布式的工作模式。可是我平时一般都是通过数据库来编程的，仅仅有分布式的文件系统还不够啊。”

“这是一个好问题。下面我来介绍一下基于 HDFS 的数据库系统。”

与传统的文件系统相似，HDFS 解决了数据存储的基本问题。可是，作为文件系统，HDFS 同样面临一个问题：缺乏良好的数据组织和访问，对于开发大型应用而言，这实在是不太方便，因此需要一个类似传统关系型数据库的管理系统。在此大环境下，Apache HBase (<http://hbase.apache.org/>) 应运而生了。HBase 是 Apache 的 Hadoop 项目的子项目，当前最新版本是 1.1.2。它是一个分布式的、面向列的开源数据库，该技术的开发同样是受 Google 的一篇论文所启发，即《Bigtable：一个结构化数据的分布式存储系统》（“Bigtable: A Distributed Storage System for Structured Data”）。Bigtable 利用了 Google 文件系统 GFS 来提供分布式数据存储，类似地，HBase 是在 Hadoop 的 HDFS 基础之上提供了 Bigtable 的能力。Hadoop 和 Database 两个英文单词的叠加也是 HBase 英文名称的由来。HBase 不同于一般的关系数据库，它是一个适合于非结构化数据的数据库，最大的特点是基于列而不是基于行的模式进行存储。那么，HBase 为什么要选择这样的 NoSQL 的设计方式呢？如此的设



计又能带来哪些好处呢？为了更好地理解这些，我们先来简单回顾一下关系型 SQL 数据库的背景。

在本章伊始，我们就已经介绍了关系型数据库和其核心元素 ER 图。在实际的系统实现中，关系模型都是通过二维表格来表示的，一个关系型数据库就是由若干个二维表格和它们之间的联系所组成的。借用图 3-1 的 ER 图，可以设计出如图 3-3 所示的二维表。

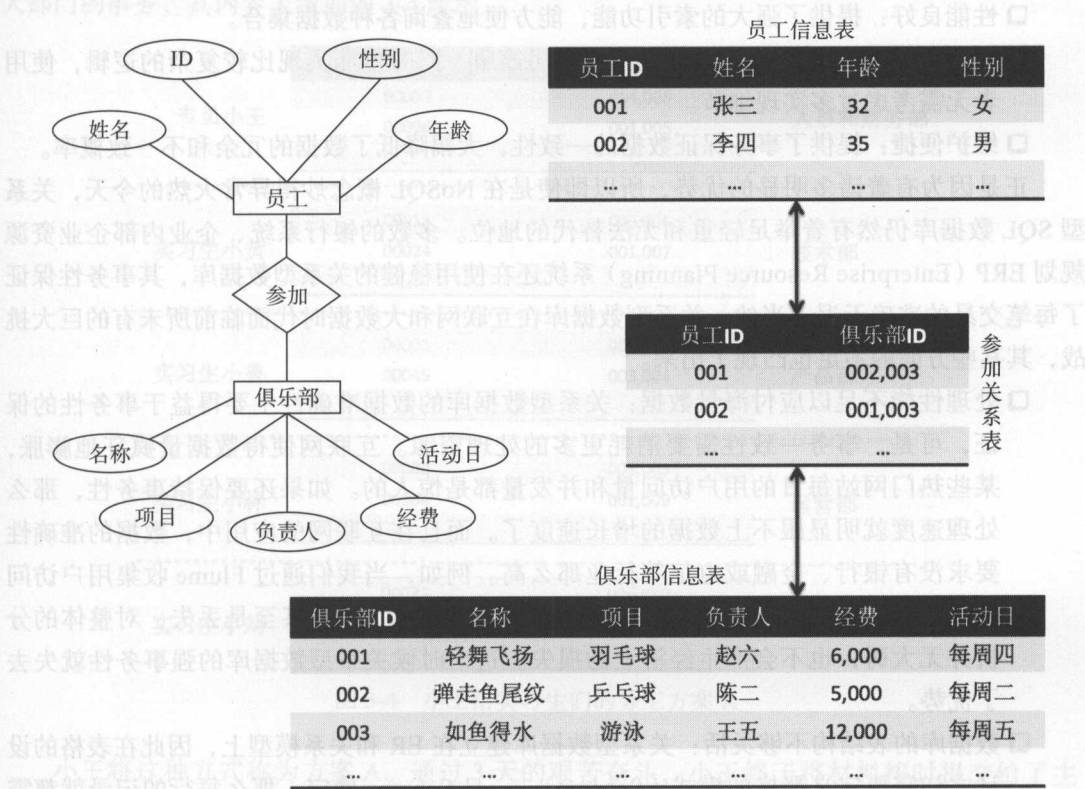


图 3-3 员工和俱乐部的二维表设计

其中，俱乐部 ID 在现实场景中可能并不存在，但是由于处理的需要，一般数据库系统会自动添加这个属性。除此以外，每张表格的列都对应 ER 图中实体的一个属性，例如员工信息表中就有员工 ID、姓名、年龄和性别共 4 个属性，而俱乐部表格中的列也包括了名称、项目、负责人、经费和活动日的属性。在数据库中这些列被称为“字段”。表的每一行则代表一个实体，例如员工信息表的第一行代表员工张三，而俱乐部信息表的第三行代表“如鱼得水”游泳俱乐部。在数据库中这些行被称为“记录”。而我们可以通过增加第三张表格（参加关系表）来体现员工和俱乐部之间的关系，从表中可以看出，001 号员工张三参加了乒乓球和游泳俱乐部。如果需要使用 SQL 语言来针对这些表格进行查询，也是非常方便的，下面列出几个最基本的使用案例：



SELECT 姓名 FROM 员工信息表

含义：返回“员工信息表”中所有员工的姓名

INSERT INTO 俱乐部信息表 VALUES (“夸父追日”，“慢跑”，“吴九”，“3,000”，“每周六”)

含义：建立新的“慢跑”俱乐部

不难发现，这种关系数据库有着非常明显的优势：

- 理解容易：相信大家从图 3-3 中已经看出来，二维表非常贴近人类的思维逻辑。
- 性能良好：提供了强大的索引功能，能方便地查询各种数据集合。
- 使用方便：SQL 结构化查询语言，入门简单，同时也能实现比较复杂的逻辑，使用者无需考虑过多实现细节。
- 维护便捷：提供了事务保证数据的一致性，大幅降低了数据的冗余和不一致概率。

正是因为有着诸多明显的优势，所以即使是在 NoSQL 概念炒得异常火热的今天，关系型 SQL 数据库仍然有着举足轻重和无法替代的地位。多数的银行系统、企业内部企业资源规划 ERP (Enterprise Resource Planning) 系统还在使用稳健的关系型数据库，其事务性保证了每笔交易的准确无误。当然，关系型数据库在互联网和大数据时代面临前所未有的巨大挑战，其某些方面的不足也凸现了出来。

- 处理性能不足以应付海量数据：关系型数据库的数据准确性主要得益于事务性的保证，可是，事务一致性需要消耗更多的处理资源。互联网使得数据量疯狂地膨胀，某些热门网站每日的用户访问量和并发量都是惊人的。如果还要保持事务性，那么处理速度就明显跟不上数据的增长速度了。而且在互联网的应用中，数据的准确性要求没有银行、金融或电信等行业那么高。例如，当我们通过 Flume 收集用户访问网站的行为数据时，在某个时间点的点击记录发生了延迟甚至是丢失，对整体的分析并无大碍，也不会产生经济上的损失。这个时候关系型数据库的强事务性就失去了优势。
- 数据库的表结构不够灵活：关系型数据库建立在 ER 和关系模型上，因此在表格的设计之初需要定义严格的模式 (Schema)。一旦 Schema 确定，那么每行的记录就都需要严格遵守这个规定。万一需要修改，整个过程也是比较复杂的。修改完毕后，既有的所有记录都要根据这个新的 Schema 做出相应的调整。然而，互联网领域强调的就是“变化”。每时每刻创新的想法都在诞生，项目的进度走的都是敏捷迭代方式，那么数据的定义也不可能一成不变。频繁的改动会使得关系型数据库疲于应付。

由于不能很好地适应互联网时代的数据处理需求，人们开始设计 NoSQL 的方案以作为补充。针对上述两个主要的不足，HBase 首先集成了 Hadoop 的 HDFS 文件系统<sup>①</sup>，用于提供可水平扩展的能力，为大规模数据量做好了准备。另外，HBase 还提供了非常灵活的列式存储，这是有别于关系型数据库行式存储的方式。关于列式和行式存储的差异，可通过公司俱乐部的案例来做进一步说明。小王是某大公司的人力资源专员，她有一项重要的任务是

① HBase 也可以不用建构在 HDFS 之上，不过这样就无法发挥 HDFS 的并行处理能力了。

负责员工的福利事宜，也包括员工的业余爱好俱乐部。一日，主管让小王将全公司员工所参加的俱乐部情况统计一遍，3天之内完成。非常遗憾的是，这家公司还没有将这些信息输入ERP系统，全公司10 000多名员工的俱乐部资料，全部需要小王手工整理出来。即使加班加点，对她而言3天完成也是不可能的。咋办呢？小王只好向老板申请增加几名实习生，对于实习生，她是这么安排的：按照公司的部门来划分，4个实习生加上自己共同处理公司五大部门的事务，其内容大致如图3-4所示。

	员工ID	俱乐部ID	
专员小王	00037	004,008	人事和财务部
	00096	001,005	
	...	...	
-----			
实习生小黄	00018	005,006	技术部
	00024	001,007	
	...	...	
-----			
实习生小鲁	00001	002,003	产品部
	00045	003,004	
	...	...	
-----			
实习生小林	00002	001,003	运营部
	00104	001,009	
	...	...	
-----			
实习生小刘	00073	006,010	市场部
	00098	008,011	
	...	...	

图 3-4 小王给实习生们的分工方案 A

小王将这种方式称为方案 A。通过3天的艰苦奋斗，小王终于将材料按时提交给了主管。主管看过后非常满意，小王对自己工作安排的合理性也很是得意。可是，没过几天，主管又提出了新的需求：她想知道每位员工参加俱乐部活动的出席率如何。于是小王和每位实习生只能再次统计一遍。又过了几天，主管提出她还想知道每位员工参加俱乐部比赛后，获得名次的情况。新的需求不断增加，每次小王和实习生都非常辛苦。而且一旦员工参加了新的俱乐部，或者是出席率发生了变化，数据更新工作又将是无法避免的。小王开始思考，这样的分工真的是合理的吗？有没有可能换一种方式？她发现主要的变化大多是新增的统计项目，而且新增的项目也不一定适合所有员工，例如俱乐部比赛名次，有些俱乐部根本不组织比赛，也无所谓比赛名次了。那么，如果让每个实习生专职负责若干统计项目，工作效率是否会更高呢？例如根据图3-5的形式来分工。

这次，小王负责统计员工参加了哪些俱乐部，而其他4位实习生分别统计出席率、比赛名次、赞助经费和俱乐部内的职务。如果再有新的项目需要统计，小王也分配给某人专

职来维护。小王将其称为方案 B。相对于方案 A，方案 B 的好处在于，如果只是新增或更新某一项数据，只需要一位人员来操作，而其他人完全可以不用理会。如果将来这些数据不再是人工操作，而小王和伙伴们的工作也交由计算机来处理的话，那么 A 方案对应就是行式存储，而 B 方案对应的就是列式存储。两者孰优孰劣并无定论，而是要看具体的应用场合。刚刚提到小王的主管提出的新需求很多，经常需要增加统计项目，这就对应于二维表中列的维护，因此适合列式存储。再假设一下，主管没有那么多需求，但是经常有新员工入职，每位员工对应于二维表中的一行，那么这就会涉及行的维护，此时行存储就更适合。如果这个时候还采用列存储，就意味着每次新增员工，所有的小伙伴们都要修改手头的表格。

	小王	小黄	小鲁	小林	小刘
员工ID	俱乐部ID	出席率	比赛名次	赞助经费	职务
00037	004,008	20%	/	200	/
00096	001,005	25%	2	300	/
...	...	...	...	...	...
00018	005,006	6%	5	200	/
00024	001,007	8%	2	200	裁判
...	...	...	...	...	...
00001	002,003	35%	/	300	/
00045	003,004	25%	5	100	/
...	...	...	...	...	...
00002	001,003	8%	1	100	教练
00104	001,009	24%	4	200	/
...	...	...	...	...	...
00073	006,010	30%	3	300	裁判
00098	008,011	56%	/	300	/
...	...	...	...	...	...

图 3-5 小王新的分工方式方案 B

在了解列式存储相对于行式存储的优势之后，我们就能明白为什么列式存储更适合互联网灵活多变的环境了，它并不要求开发者在起初就给出完美的数据 Schema 定义，而是允许在随后的进展过程中不断被优化，定义修改所导致的历史数据的更新成本也会更小。接下来看看 HBase 使用了怎样的数据模型来实现列式存储。下面首先列出几个关键的概念：

- ❑ 表格 (Table)：HBase 同样用二维表格来组织数据。
- ❑ 行 (Row)：在表格里，每一行代表一条记录，这与关系型数据库一致。每行通过行键 (Row Key) 进行唯一标识。
- ❑ 列族 (Column Family)：了解这点很关键，行里的字段按照列族进行分组，可以看作一堆属性或字段的集合。列族的定义决定了 HBase 数据的物理存放。因此，列族需要预先定义，而且不要轻易修改。每行都拥有相同的列族，不过 HBase 并不要求每



个列族都存储数据。这也是为了满足灵活的数据定义需求。

❑ 列限定符 (Column Qualifier): 列族里包括多个属性, 限定符可以帮助定位列族里的数据。与列族不同, 列限定符没有必要预先被定义, 因此每行可以拥有不同数量和名称的限定符。图 3-5 中的表格存在很多 “/”(空缺值), 对于这样的表格, 灵活的列限定符可以减少不必要的存储, 提升处理稀疏矩阵的能力。

❑ 单元 (Cell): 二维表里的单元格, 通过行键、列族和列限定符来唯一确定。存储在其中的值称为单元值 (Cell Value)。

❑ 版本 (Version): 注意, 这是 HBase 与很多数据库的不同之处。即使单元被确定了, 里面的单元值仍然可以根据时间的不同拥有多个版本。版本用时间戳 (Timestamp) 来标识。读取的时候如果没有指定时间戳, 那就默认获取最近的版本。

如果将行键和列限定符对应于关系型数据库的行和列, 那么 HBase 主要就多了列族和版本。记住这 6 个主要概念, 理解 HBase 读取数据的机制就并不困难了。从图 3-6 中可以看出 HBase 的坐标体系。其中最有趣的地方在于, HBase 中可以不用提供全部坐标。如果只提供行键, 那么就会返回某行的整行。如果提供行键、列族和列限定符, 那么就返回某行某列的最新单元值。再进一步, 如果同时提供了行键、列族、列限定符和时间戳, 那么就返回某行某列单元值的某个版本。

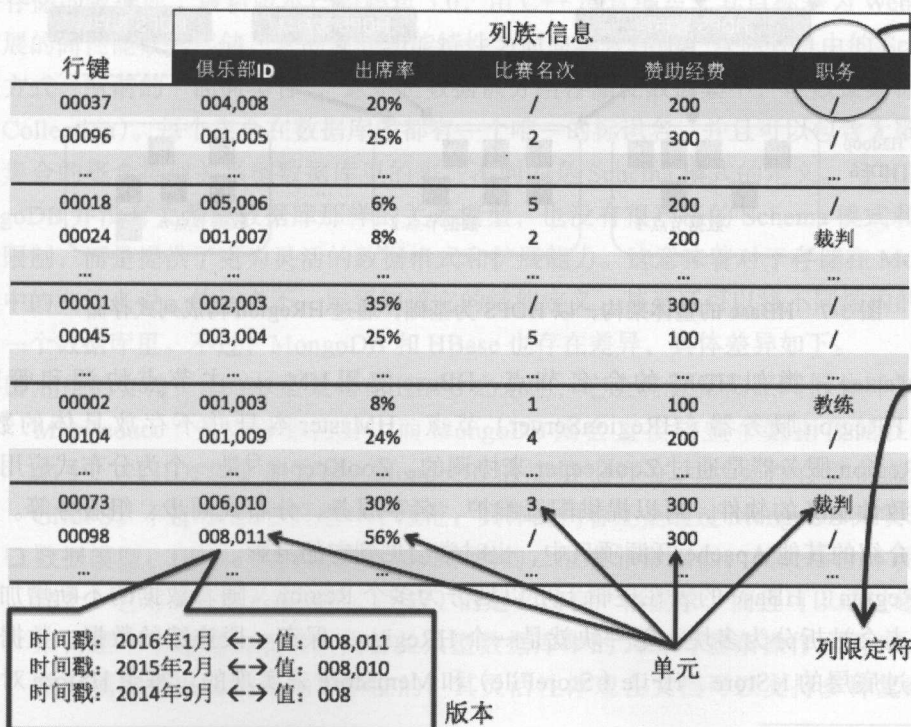


图 3-6 HBase 的主要概念: 行键、列族、列限定符、单元和版本



其中，如果给定行键 00096，将返回如下信息：  
{ 俱乐部 ID：“001,005”，出席率：“25%”，比赛名次：“2”，赞助经费：“300” }  
如果给定行键 00096、列族“信息”和列限定符“出席率”，那么返回出席率：“25%”。  
如果给定行键 00098、列族“信息”、列限定符“俱乐部 ID”和时间戳“2015 年 2 月”，那么返回值是“008,010”。

了解完 HBase 的基本概念和数据模型，再来看看它的体系架构，如图 3-7 所示。为了保证良好的扩充性和并行处理能力，HBase 是架构在 Hadoop 的 HDFS 上的。此外，它通过 HRegion 和 HStore 来实现列族的存储。具体来说，其中的主要元素如下。

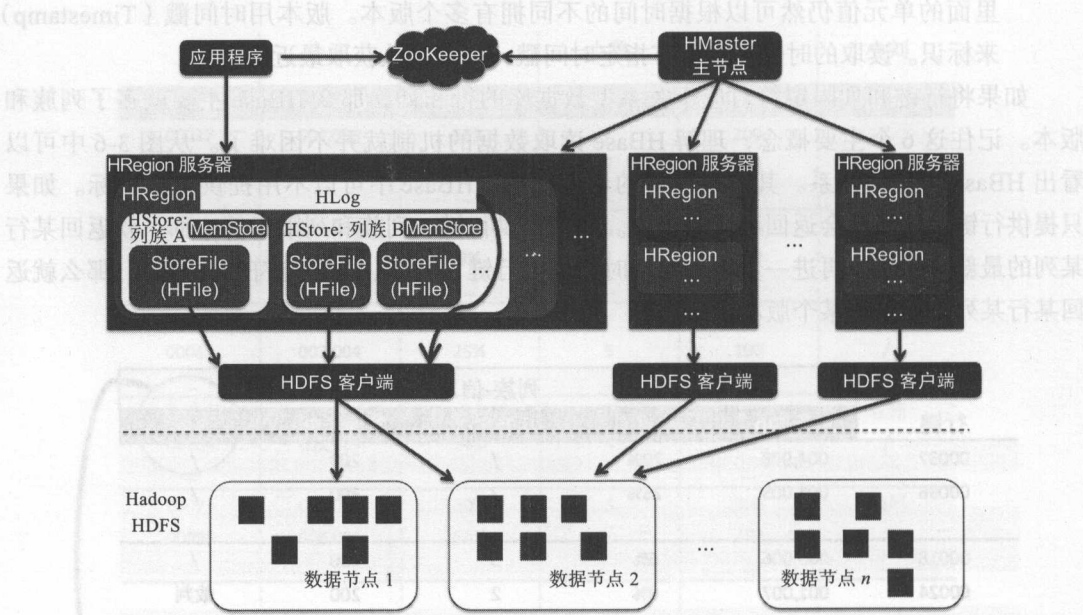


图 3-7 HBase 的整体架构，以 HDFS 为基础，通过 HRegion 构成列式存储

- ❑ HMaster：类似 HDFS 的命名节点，HBase 使用 HMaster 主节点协调和管理多个 HRegion 服务器（HRegionServer）节点。HMaster 本身并不存放具体的数据。HRegion 服务器是通过 ZooKeeper 来协调的。ZooKeeper<sup>①</sup>是一个为分布式应用提供一致性服务的软件，可以提供配置维护、名字服务、分布式同步、组服务等。在后面介绍的其他 Apache 开源项目中，也时常会见到它的身影。
- ❑ HRegion：HBase 的表在逻辑上可以划分为多个 Region。随着数据的不断增加，一张表会被拆分为多块。每一块就是一个 HRegion，保存一段连续的数据。数据都是通过底层的 HStore、HFile（StoreFile）和 MemStore 来实现的。每个 HStore 对应于

① ZooKeeper 的名字来源是因为它管理的很多分布式系统喜欢使用动物来代言，“动物园的管理者”再形象不过了。

一个列族，HFile 和 MemStore 分别是文件和内存的存储。此外，HRegion 中还包含 HLog 来记录日志以便于事故后的恢复。

❑ HRegion 服务器 (HRegionServer): 多个 HRegion 由 HRegion 服务器来管理。

HBase 的列存储设计为灵活的表结构提供了基础，在实际应用中修改列族的定义是很常见的。对于新的业务需求，不断增加列族或限定符也是不错的选择，从二维表的视角上看，这种模式会导致表列越来越多，越来越宽，因此我们也可以形象地将其称为“宽表”。

“看来 HBase 的宽表模式，既可以节省关系型数据库中的连接操作或存储空间，同时还能将不同 Schema 模式的数据进行混合。是不是可以将其运用在异构数据源的统一和集成上？我们公司即将进行的线下业务，每个商铺都有自己的 ERP 系统，数据格式都不尽相同，我正头疼这个事情呢。”

“你的理解完全正确，可以通过 HBase 进行商铺历史数据的整合，而且还不用担心水平的扩展性”。

### 3.1.3 MongoDB

除了 HBase 这样的列式存储，NoSQL 的数据库中还有一部分采用的是文档型存储，MongoDB 就是其中的一个典型代表。MongoDB (<https://www.mongodb.org/>) 是基于分布式文件存储的数据库，最新版本已经超过 3.0，由 C++ 语言编写，其目标是为 Web 应用提供可扩展的高性能数据存储。它的主要功能特性为面向集合存储，且拥有自由的 Schema 模式定义方式。所谓的“面向集合”，意思是数据被分组存储在数据集中，该数据集称为一个集合 (Collection)。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库里的表，不同的是 Schema 模式的定义。与 HBase 类似，MongoDB 并没有关系型数据库那样的关系模型，也没有很严格的 Schema 模式和数据一致性的限制，而是提供了更为灵活的数据格式和扩展能力。这意味着对于存储在 MongoDB 数据库中的文件来说，使用者不一定要知道它的结构定义，甚至还可以将不同结构的文件存储在同一个数据库里。不过，MongoDB 和 HBase 也存在差异，具体差异如下。

❑ 底层的支持：HBase 是架构在 HDFS 之上的，它会将数据按照列拆分开来存储，对 MapReduce<sup>①</sup> 支持得非常好。而 MongoDB 则会直接将整个数据存储 in 文件系统之上，与 HDFS 没有关联。MongoDB 的底层也存在大型文件存储的概念，称为 GridFS。不过 GridFS 只是一个规范，具体的内容还是通过 MongoDB 来实现的。

❑ 数据模型：HBase 的数据模型和关系型的二维表非常相似，其灵活性体现在列式存储上，它对列（或者说属性、字段）的定义没有严格要求，而且可以通过超多的列族来构建一个超宽的表格，代替关系型数据库中的 Join 等复杂操作。而 MongoDB 的数据模型是以文档为基本单位的，其灵活性体现在文档可以支持多种复杂的结构。

① Hadoop MapReduce 的内容会在第 4 章数据处理中做具体介绍。

在 IT 领域，“文档”本身就是一个宽泛的概念，在本书中也会多次提及（在第 5 章的信息检索部分还会提及，基本概念相当，但是具体内容和此处稍有不同）。这里的文档可以简单地理解为一堆属性或字段的集合，与二维表中的行相对应，不过最大的区别在于，MongoDB 文档中的属性或字段并不局限于特定的类型，如第一篇文档的“俱乐部”属性可以是一个字符串，而第二篇文档的“俱乐部”属性就变成了一个新的复合对象，包括俱乐部的名称、成立时间、负责人等。这在关系型数据库中一般都是通过 Join 来实现的，而 HBase 需要做的就是将嵌套的内容转化成扁平的列族。图 3-8 展示了关系型数据表、HBase 的宽表和 MongoDB 文档表的对比情况。

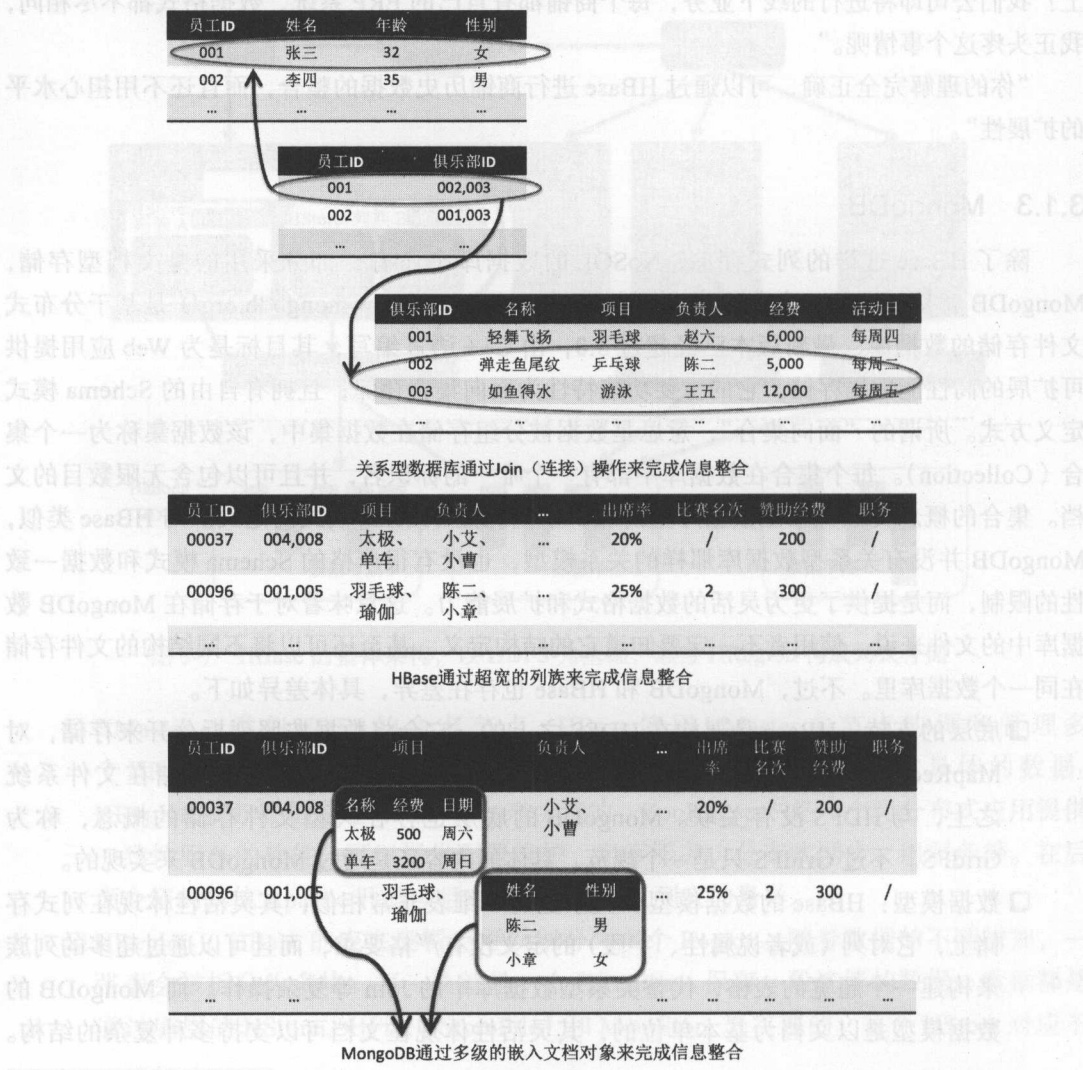


图 3-8 关系表、HBase 和 MongoDB 的数据模型对比



□对数据的拆分：HBase将列族进行拆分，分布在HDFS的不同资源节点上。MongoDB的设计者可能考虑到了嵌入多级文档的复杂性，对文档的大小做了限制，而且不会将单个文档的数据再进行切分和分布式存储，这点更接近行式存储。不过，文档的灵活定义仍然保证了对非关系型数据的支持。

除了面向文档和模式更灵活，MongoDB最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系型数据库单表查询的绝大部分功能，而且还支持对任意字段的数据建立多级索引（还包括基于地理空间的索引），能够大幅地提升处理效率。当然，即使没有架构于HDFS和Hadoop生态之上，MongoDB仍然提供了良好的并行处理和高扩展性，支持数据的复制和恢复，实现了故障恢复等功能。

## 3.2 非持久化存储

“好了，大宝，关于持久化存储，我们先介绍到这里。无论是Hadoop的HDFS、HBase还是MongoDB等其他非关系型数据库，它们的目标都是为了解决大量数据的高效分布式存储。虽然具体的实现和应用场景有所不同，但是简而言之这些都是支持持久化的存储，强调的是将数据永久记录在磁盘上。”

“小明哥，你在一开始还提到了非持久化的方式，那是什么呢？”

“对，还有一种存储是在内存中的非持久化存储，通常其所能承载的数据量不如磁盘那么大，在断电后数据也会消失，但是其明显的优势在于读取速度非常快。在这种非持久化的存储方式中，大多数采用的是散列（Hash）的Key-Value存储方式，适合需要用到缓存（Cache）的场景。”

“那缓存和散列又是什么？”

“我先介绍一下缓存和散列的概念。”

### 3.2.1 缓存和散列

缓存（Cache）可以被看作计算机系统的伟大发明之一，它的应用在该领域中是普遍存在的。小到计算机的中央处理器（CPU）、主板、显卡等硬件，大到大规模的互联网站点，都在广泛使用缓存并从中受益。举例来说，自己组装过个人计算机硬件的朋友可能会更关心中央处理器、主板的缓存有多大；如果是电子游戏的粉丝，还会关注显卡的缓存有多大。这些硬件所采用的缓存是比普通的内存更快的介质，可以显著提升计算机的运行速度。不过，考虑到个人计算机的成本和设计框架，这里的高速缓存容量不可能很大，若干兆字节（MB）是比较常见的设置。而在网站的架构设计中，一般不会像个人计算机样采用高速的缓存介质，普通的服务器内存就已足矣，但是容量可以更大，至少是数个吉字节<sup>①</sup>（GB）。此处将

① 一个吉字节=1,024兆字节。



这个概念抽象出来，将缓存定义为数据交换的缓冲区，它的读取速度远远高于普通存储介质，作用是帮助系统更快地运行。当某个应用需要读取数据时，会优先从缓存中查找所需要的内容，如果找到了则直接获取，这个效率比读取普通存储的效率更高。如果缓存中没有发现所需要的内容，再到普通存储中寻找。

在了解缓存的概念之后，我们看看缓存的几个要素。

□ **硬件性能**：缓存的普遍规律是以高速读取介质来充当相对低速的介质的缓冲。由于缓存的应用场景非常广泛，因此没有绝对的条件来定义何种性能可以达到缓存的资格。例如，在个人计算机系统中，内嵌于主板中的一级（L1 Cache）和二级（L2 Cache）高速缓存，可以存放内存条中常用的数据，以便提升内存读取速度，这时它就充当了内存的缓存角色。同样，相对于硬盘而言，内存条又可以充当硬盘的缓存，存放硬盘读取中常用的数据，用于提升硬盘读取的速度。

□ **命中率**：缓存之所以能提升访问速度，主要是因为能直接从高速介质中读取，这种情况称为“命中”（hit）。但是，高速介质的成本是非常昂贵的，而且一般也不支持持久化存储，因此放入数据的容量必须受到限制，只能是全局信息的一部分。那么，一定是有部分数据无法在缓存中读取，而必须要到原始的存储中查找，这种情况称之为“错过”（missed）。可以通过公式（3-1）来简单地定义命中率。

$$HitRatio = \frac{|H|}{|V|} \quad (3-1)$$

其中  $|V|$  是整体的数据访问次数，而  $|H|$  是能够在缓存中查找到数据的次数。如果命中率高，系统能够频繁地获取已经在缓存中驻留的数据，速度就会明显提升。如果获取的命中率非常低，系统仍然需要到低速介质上进行读取，那就不可能带来性能的提升。那么接下来的问题就是，如何在缓存容量有限的情况下，尽可能地提升命中率呢？人们开始研究缓存的淘汰算法，通过某种机制将缓存中可能无用的数据剔除，然后向剔除后空余的空间中补充将来可能会访问的数据。最基本的策略包括最少使用（Least Frequently Used, LFU）和最久未用（Least Recently Used, LRU）。LFU 会记录每个缓存对象被使用的频率，并将使用次数最少的对象剔除。而 LRU 会记录每个缓存对象最近使用的时间，并将剔除使用时间点最久远的对象。

□ **更新周期**：缓存之所以处理效率惊人，除了硬件性能上的先天优势外，还需要保证较高的命中率。但是，被访问的数据不会一成不变，对于变化速度很快的数据，我们需要将变动主动更新到缓存中，或者让原有内容失效，否则用户将读取到过时的内容。在无法及时更新数据的情况下，高命中率反而变成了坏事，轻则影响用户交互的体验，重则会导致应用逻辑的错误。

□ **应用场景**：由于缓存应用很广泛，要一一细分不太容易，因此只能从大体上简单地将它们分为系统级和应用级。系统级缓存是指操作系统提供的缓存，主要是利用主

板缓存、显卡缓存和内存等提升操作系统的运行速度，对于普通使用者或开发者而言基本上都是透明的。而应用级缓存则和开发者的架构设计有关，例如一个在线用户量很大的网站，随着数据量的增大、访问的集中，就会出现后台持久化存储负担加重、数据库响应恶化、网站显示延迟等重大影响。而缓存就成为解决或优化这些性能问题的关键。本章后面提到的非持久化缓存，都是与应用级缓存相关的话题。

图 3-9 体现了缓存设计的因素和大致的工作流程。

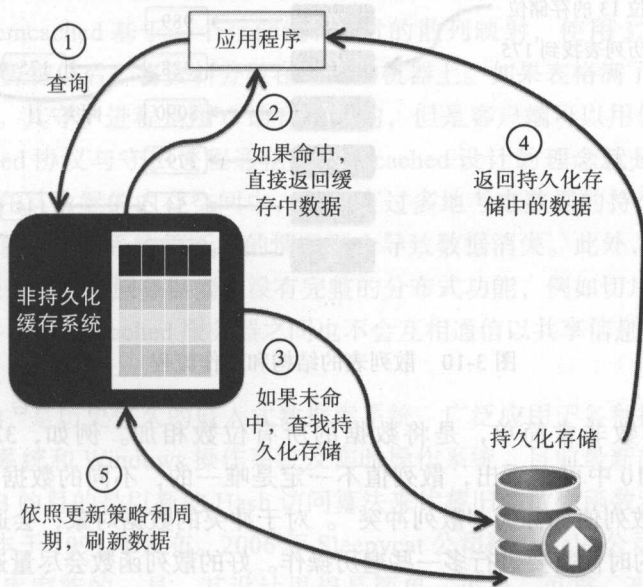


图 3-9 缓存的工作流程

了解这些要素之后，我们不禁要问，在实际运用中是如何实现缓存的机制的呢？这里就需要提到散列（Hash）和散列表（Hash Table）的概念了。曾经有人提出，假如在计算机编程语言的世界里，只能存在一种数据结构，那么你会选择什么？大部分“程序猿”和“攻城师”都会选择散列表。虽然只是说笑之词，不过也足以说明散列结构在计算机领域举足轻重。在前面介绍持久化存储时，无论是 HBase 还是 MongoDB 都要用到散列的设计理念。不过，对于非持久化的缓存而言，其设计和研发很多都是围绕散列展开的，这个概念显得更加重要了。

散列（或散列函数）通过一定的算法将原始的数据转换为一个唯一（或者尽可能唯一）的数值，这个数值可以称为散列值。散列表，也称为散列映射，写入的时候以数据的散列值作为键（Key），以待写入的数据作为值（Value），进行 Key-Value 配对的存储。这样可以带来一个非常明显的好处，对于给定的数据，我们可以通过散列值计算快速定位，以加快查找的速度。无论散列表中有多少数据，插入和删除操作只需要耗费接近常量的时间，即

O (1) 的时间复杂度<sup>Ⓔ</sup>，这正好满足了缓存高速运作的需求。图 3-10 展示了散列表工作的基本流程。

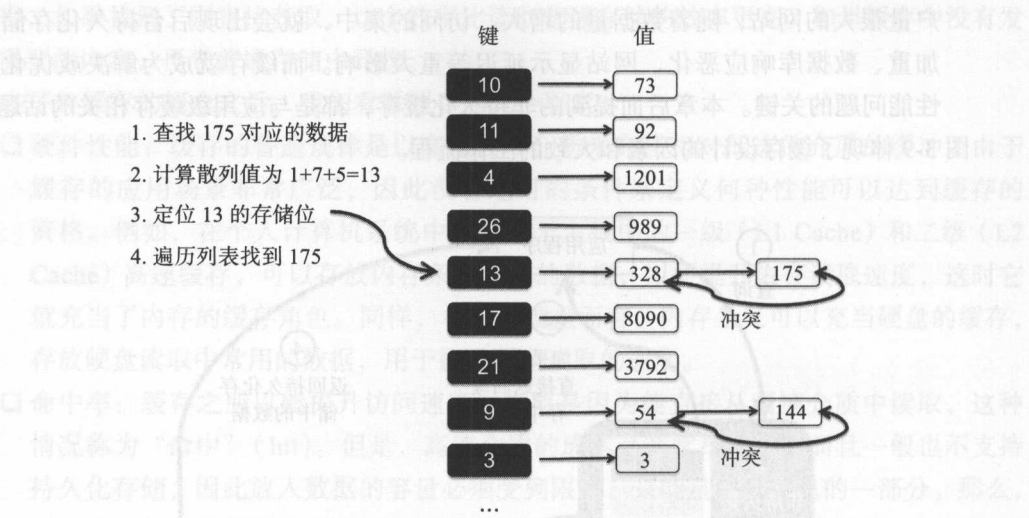


图 3-10 散列表的结构和工作流程

这里的散列函数非常简单，是将数据的所有位数相加。例如，328 的散列值就是 3+2+8=13。从图 3-10 中可以看出，散列值不一定是唯一的，不同的数据经过散列函数后，可能会得出一样的散列值，称为“散列冲突”。对于冲突的数据对象，会通过列表存储所有对应的数据，查询的时候需要进行多一些遍历操作。好的散列函数会尽量避免这种冲突。即使存在冲突，通过散列寻址还是可以大幅提升查询效率的。

“哦，这下我明白了缓存的意思，也知道如何通过散列来实现最基础的缓存机制了。那么是不是可以运用这层缓存来减轻数据库的负载呢？”

“思路很对，尤其是对你们这种电商的业务模式，对于缓存的需要更为迫切。当商品数量和用户访问流量达到一定的规模之后，如果每次都要去数据库获取实时的库存、价格、订单等信息，性能会很容易下降，系统甚至会有崩溃的风险。这个时候完全可以采用缓存来解决，将商品的 ID 作为键，而相关的库存、价格等作为值，这样最常用的商品信息都可以直接放入内存，读写速度将大大提升。”

“那么，有哪些非持久化存储系统可以支持缓存呢？”

“嗯，下面我们分别介绍应用级的几种缓存系统，包括 Memcached、Berkeley DB 和 Redis。不过，需要注意的是，系统属于持久化还是非持久化，有时划分得并没有那么绝对。目前很多系统在设计的时候都会支持多种功能。例如，我们也可以使用 MongoDB 做缓存，而下面即将介绍的缓存方案 Redis 和 Berkeley DB 也是支持持久化存储的。这里的划分主要

Ⓔ 时间复杂度在第 7 章的效能评估中有具体介绍。这里可以简单地理解为时间上的开销。



是按照这些系统目前在业界最常见的应用方式进行的。”

### 3.2.2 Memcached 和 Berkeley DB 简介

Memcached (<http://memcached.org/>) 是一个高性能的分布式内存对象缓存系统, 目前最新版本更新到了 1.4, 用于高性能 Web 应用以减轻底层存储的负载。最初 Memcached 是 Danga Interactive 为了 LiveJournal 所研发的, 后来被许多软件所使用, 成为一套开放源代码的软件, 以 BSD License 授权协议发布。它通过在内存中缓存数据和对象来减少读取数据库的次数。Memcached 基于一个存储键/值对的散列映射, 使用 32 位元的循环冗余校验 (CRC-32) 计算键值后, 将资料分散在不同的机器上。如果表格满了, 则通过 LRU 机制将旧的数据剔除。其守护进程是用 C 语言编写的, 但是客户端可以用任何语言来开发, 然后通过 Memcached 协议与守护进程通信。Memcached 设计的理念就是要简单易用, 保存的数据都被存储在其内置的内存空间中, 并没有过多地考虑数据的持久性问题。因此在重启 Memcached、重启操作系统等断电的情况下会导致数据消失。此外, 尽管 Memcached 是“分布式”缓存服务器, 但服务器端并没有完整的分布式功能, 例如切片 (Sharding) 和副本 (Replication), 各个 Memcached 服务器之间也不会互相通信以共享信息, 因此需要应用端来实现类似的逻辑。

Berkeley DB<sup>①</sup> 是历史悠久的嵌入式数据库系统, 广泛应用于各种操作系统, 包括大多数 UNIX 类操作系统和 Windows 操作系统及实时操作系统, 目前最新的版本是 12.1。最初开发 Berkeley DB 的目的是以新的 Hash 访问算法来代替旧的散列函数和大量的数据库操作实现, 第一个版本于 1991 年发布。2006 年 Sleepycat 公司被 Oracle 公司收购, Berkeley DB 成为 Oracle 数据库家族的一员。其设计思想是简单、小巧、可靠、高性能。Berkeley DB 可以保存任意类型的键/值对, 而且可以为一个键保存多个数据。它的函数库可能只有几兆字节, 但是却能管理数百 TB 的数据, 同时支持数千的并发线程操作。尽管架构很简单, Berkeley DB 却支持很多高级的数据库特性, 比如 ACID 数据库事务处理、细粒度锁、XA 接口、热备份及同步复制等。与 Memcached 相比, Berkeley DB 更为轻量级, 核心部分不支持分布式, 但是它支持持久化的功能, 这是 Memcached 所不具备的。

### 3.2.3 Redis 简介

Redis (<http://redis.io/>) 的全称是远程字典服务器 (REmote DIctionary Server), 它是一个开源的、高性能的、基于键-值型的缓存和存储系统, 提供了多种编程语言的 API 接口, 近几年逐步开始流行于业界。2008 年, 意大利的一家创业公司为了满足业务需求, 开始量身定制一套数据库系统。在其基础上, 2009 年首个 Redis 版本发布。2010 年年初, Redis 的开发工作开始由 VMware 赞助。时至今日, 最新的版本已经更新到 3.0.5, 支持分布式集群。

① <http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/downloads/index.html>



Redis 的特性主要包括：提供了极高的性能、支持多种数据类型、支持事务性、可设定生命周期、提供持久化存储等。

❑ 超高的性能：Redis 的处理速度非常快，不少数据显示它可以每秒进行数十万次的操作。

❑ 支持多种数据类型：这是 Memcached 所不具备的。除了最基础的字符串，Redis 还支持多数开发人员常用的散列（Hash）表、列表（List）、集合（Set）、有序集合（Sorted Set）。丰富的数据结构选择，使得设计者们可以更容易地解决多样的业务需求。这点与 MongoDB 对复杂文档对象的支持是颇为相似的。

- 字符串是 Redis 中最基本的数据类型，能够存储任何形式的字符串，包括二进制数据。设计者可用其存放用户的姓名、年龄、照片等。

- 散列类型略微复杂一点。其实是在值里再存放一个散列表结构，类似于多层嵌套的概念。不过，这个内嵌的散列表的值只能是字符串，不能是其他复杂类型。这种类型的好处是实现了编程对象和属性（字段）的映射，更符合面向对象的设计理念。

- 列表类型可以存储一个有序的字符串列表，并且支持向列表两端添加元素、获取部分元素等操作。

- 集合的概念与在数学里的概念相同，其中存放的数据不能重复，并且是无序的状态。相比之下，列表中的数据是可以重复，并且是有序的。Redis 强大的功能也能直接对集合进行取交集和取并集等操作。

- 有序集合和集合类似，但是存放的数据是可以保持顺序的。

❑ 支持事务性：事务性是经典数据库的一个重要特性，银行转账是其最常见的应用案例。当用户 A 转账给用户 B 时，首先是从 A 账户上扣取一定的金额，然后在 B 账户上增加同样的金额。如果 B 账户未能增加这笔钱，那么必须将 A 账户所扣取的部分一厘不少地补还，这就是事务的基本概念。Redis 通过一套命令的组合，在一定程度上实现了事务性。之所以说“一定程度”，是因为 Redis 只能保证这些命令要么都执行，要么都不执行。它并没有考虑执行过程中可能出现的错误，也没有为出现错误后所产生的“脏数据”提供恢复方案。

❑ 可设定生命周期：如前所述，对于缓存中变化很快的数据需要主动地将其更新，或者使缓存失效。Redis 允许开发者为键-值设置生命周期（Time To Live, TTL），然后根据这个时间，定期地自动删除已经过期的数据。这个特性非常适合于验证码、限时特惠等应用场景。

❑ 持久化存储：这也是 Memcached 暂不支持的。Redis 支持两种持久化方式，即 RDB 镜像和 AOF 日志。RDB 可以看作全量保存，符合设置的条件是，Redis 将内存中所有的数据生成副本并存储到磁盘上，日后恢复时再全部从磁盘上进行加载。而 AOF 的全称是 Append File Only，顾名思义是采用增量方式，即将每次执行的 Redis 命令记录到磁盘的 AOF 文件中，日后恢复时通过命令的日志来重构数据。从备份和容灾

的角度而言, AOF 方式备份的资源消耗更小, 而且恢复后丢失的数据也较少。不过, RDB 可能在恢复的速度上更快些。

从系统架构上来看, Redis 支持主从同步。数据可以从主服务器向任意数量的从服务器上同步, 从服务器也可以是关联其他从服务器的主服务器。这就使得 Redis 可执行树状结构的复制。由于实现了发布/订阅机制, 使得从数据库在任何地方同步数据时, 均可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的扩展性和数据冗余很有帮助。同时, Redis 还提出了“哨兵”的概念, 用于增强系统的自动化监控和故障恢复。图 3-11 展示了大体的系统框架。

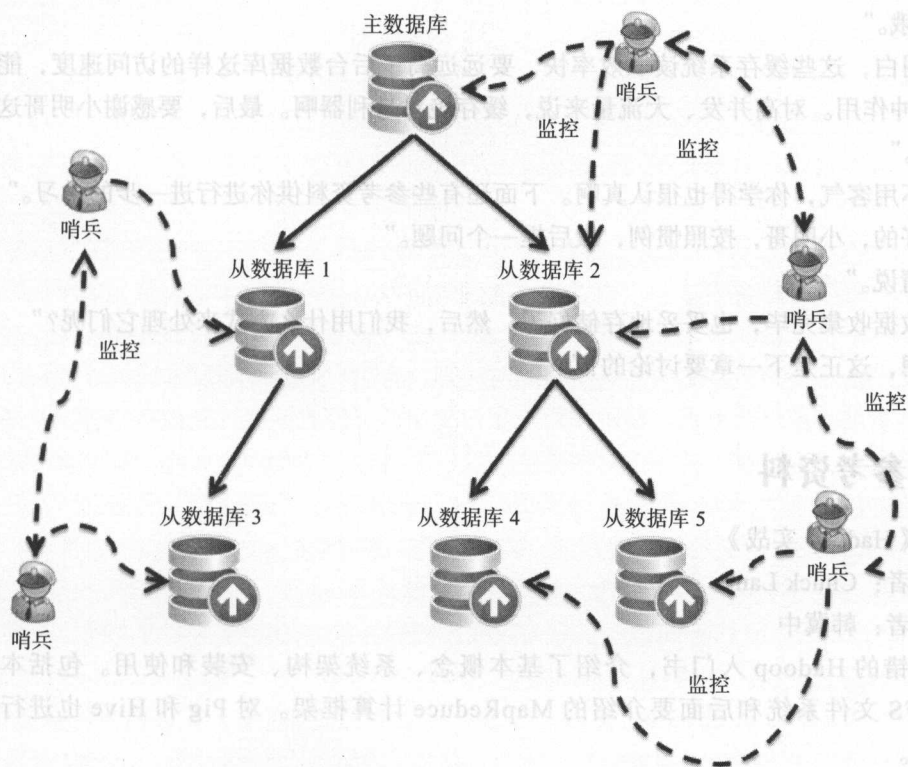


图 3-11 Redis 主从结构和哨兵监控

主从结构的 Redis 适合读多写少的场合, 对主服务器进行数据更新的写入, 对从服务器进行数据读取。但这样一来, 如果存在单点故障, 即一旦主服务器出现问题, 就无法实现数据的更新, 系统的恢复也比较麻烦。好在, Redis 3.0 之后提出了“集群”的概念, 包括预分片的技术, 对水平扩展的支持更进一步。

### 3.3 本章心得

“听了小明哥的介绍，这下不用担心收集来的数据没处放啦。传统的文件系统和关系型数据库对于普通的单机应用可能足够了，但是对于互联网的应用而言，往往数据量会太大，这时我们就需要考虑扩展性强的方案了。Hadoop 提供了一种不错的选择，其 HDFS 可以用来存储日志这种普通文件。如果需要用到数据库管理的基本功能，HBase 和 MongoDB 都是不错的选择。不过感觉 HBase 和 Hadoop、HDFS 的结合更紧密，这样入手是不是比较简单。”

“对，你可以先尝试从简单易懂的入手。别忘了，除了这些 NoSQL 的数据库可以做持久化存储，还有基于键 - 值的非持久化缓存系统也很重要，关键的时候可以极大地提升系统的性能哦。”

“明白，这些缓存系统读取效率高，要远远高于后台数据库这样的访问速度，能起到很好的缓冲作用。对高并发、大流量来说，缓存绝对是利器啊。最后，要感谢小明哥这次辛苦的讲课。”

“不用客气，你学得也很认真啊。下面还有些参考资料供你进行进一步的学习。”

“好的，小明哥，按照惯例，最后提一个问题。”

“请说。”

“数据收集完毕，也妥妥地存储好了，然后，我们用什么方式来处理它们呢？”

“嗯，这正是下一章要讨论的！”

### 3.4 参考资料

□《Hadoop 实战》

作者：Chuck Lam

译者：韩冀中

不错的 Hadoop 入门书，介绍了基本概念、系统架构、安装和使用。包括本章介绍的 HDFS 文件系统和后面要介绍的 MapReduce 计算框架。对 Pig 和 Hive 也进行了简单的介绍。

□《Hadoop 实战（第 2 版）》

作者：陆嘉恒

本书除了对 Hadoop 有了详细的介绍，还涵盖了不少 Hadoop 生态中的其他内容，包括 HBase、Hive、Pig、Mahout，以及分布式协同系统 Zookeeper，知识点覆盖比较全面。

□《HBase 实战》

作者：Nick Dimiduk, Amandeep Khurana

译者：谢磊

对 HBase 进行了比较全面的介绍，包括背景、概念、安装、操作、集群架构等。

### □《MongoDB 权威指南（第2版）》

作者：Kristina Chodorow

译者：邓强，王明辉

对 MongoDB 进行了比较全面的介绍，涵盖基础知识、核心概念、开发应用、管理和部署等多个方面。

### □《Redis 入门指南（第2版）》

作者：李子骅

Redis 技术相对比较新，这是一本不错的操作手册，介绍比较全面，适合在开发时进行参考。



## 第4章 数据处理

自从上次杨大宝提出如何处理和分析数据的疑问后，黄小明就一直在思考这个课题该如何准备。首先需要明确的是，这里的“处理”所指的范畴。第2章和第3章所提到的数据收集和存储中，有很多写入、读取的操作。从广义的概念来说，它们都可以被看作对数据的一种处理。这些都是最基础的、系统级别的处理，对用户的业务需求来说可以被认为是透明的。而本章即将探讨的处理，集中在用户如何利用现有的数据进行查询和分析，最终达到商业目标。因此，它是狭义的，针对的是应用级别的处理。正是因为如此，所以这个课题涉及的内容比之前会更为深入一些。经过了相当长时间的准备，小明找到了大宝。

“大宝，你上次说的数据处理部分，我准备了一些材料。”

“感激！从哪里开始呢？”大宝一如既往的好学。

“根据处理的及时性不同，主要分为离线处理和在线处理。介于二者之间，还会介绍提升及时性的消息机制。”小明打开了话匣子。

先来解释一下“离线”和“在线”的含义。这两个名称很形象：人们在与系统交互时，是处于在线状态（类似QQ在线）的，这个时候他们在输入任何指令后，都希望系统能非常迅速地给出响应和反馈。当他们累了，从系统下线后（类似QQ离线）要回去睡个好觉，好吧，这时候系统爱干啥干啥，一个程序运行多久我也不在乎。可以看出，在线处理就是指对实时响应要求非常高的处理，如数据库的一次查询。而离线处理就是对实时响应没有要求的处理，如批量地压缩文档。后面也会多处使用这两个概念。

### 4.1 离线批量处理

由于离线对于响应没有过高的要求，因此可以对海量数据进行批处理。Hadoop 的

MapReduce 计算是一种非常适合的框架。为了提升效率，下一代的管理框架 YARN 和更迅速的计算框架 Spark 最近几年也在逐步的成型之中。在此基础之上，人们又提出了 Hive、Pig、Impala 和 Spark SQL 等工具，进一步简化了某些常见的查询，下面分别来介绍它们。

### 4.1.1 Hadoop 的 MapReduce

前文提到过，Hadoop 在近几年逐步成为业界分布式系统的事实标准之一，其核心是 HDFS 文件系统和 MapReduce 计算框架。Doug Cutting 等人受 Google 论文《MapReduce：在大规模集群上的简化数据处理》的启发，在 Hadoop 开源项目中开始尝试实现论文所阐述的计算框架 MapReduce。在第 3 章中，我们用 Hadoop 公司存储水资源的案例，解释了 HDFS 的工作原理。名为 Hadoop 的快消品公司，专门采用 Flume 公司提供的水源来生产各式饮用水和饮料。随着市场的不断扩张，Hadoop 公司仓库的容量已经无法放下更多的原材料了，公司的高层想到了在市区周围的几个郊区再多建几处仓库，这就类似于 HDFS 的分布式存储原理。这里，再次使用 Hadoop 公司制作饮料的案例来解释 MapReduce 的原理。Hadoop 公司在解决了生产原材料的存储问题之后，开始加足马力生产各式饮料。很快，从市场的销售情况和饮用者的反馈中，公司发现了一个问题：Flume 公司提供的水源质量不太稳定，而这个不稳定性会导致某些生产出来的饮料口感较差，严重影响了自身的品牌形象和口碑。为了彻底解决这个问题，公司的产品部门提出了如下思路：

- 1) 按照 Hadoop 公司的品质定义，鉴定水源的等级，分为 1 等、2 等、3 等和 4 等。1 等水最优，4 等水最次。

- 2) 按照鉴定的等级，将 4 种水源用于不同类型的饮料生产。1 等水用于生产纯净水、2 等水用于生产矿泉水，3 等水用于生产果汁，而 4 等水则用于生产碳酸饮料。

为了做到这两点，Hadoop 公司从某自动化公司引进了最先进的自动检测仪 Mapper 和最先进的组装生产线 Reducer。Mapper 仪器可以自动对每小桶水样本进行快速检测，然后将其分为 1、2、3 和 4 等级，并贴上相应的标签。被贴上标签的小水桶们，通过传送带分别传输到相应的 Reducer，Reducer 会将同一等级的小水桶中的水源合并，汇成超级大桶用于最终的生产。图 4-1 体现了水源分级的整个流程。其中，Reducer 生产线 1 既可以生产纯净水，也可以生产矿泉水，因此被鉴定为 1 和 2 等级的水都会传送到这条组装生产线。这个过程体现的就是 MapReduce 框架的基本思想。

这种将任务切分、并行操作、最后合并结果的方法，适用于很多分布式处理的场合。例如，我们要统计大量文档集中的每个单词的词频，可以先将文章内容切分，让多个分词器同时进行统计，然后合并单词出现的次数。图 4-2 体现了统计词频的大致流程，其中 Mapper 负责将每行的文本进行切词，Reducer 负责将切出来的词进行汇总统计。还有一个常见的案例就是对于大量的数据进行去重，将不同的数据对象切分，让多个处理器根据去重时所比对的字段散列值，归拢数据并存入散列表，然后合并后除去重复的部分。图 4-3 体现了银行账务交易处理的大致流程，在该流程中，会根据账号进行去重，Mapper 负责将交易记

录中的顾客账号部分提取出来，而 Reducer 负责将同样的账号归并在一起以除去冗余。

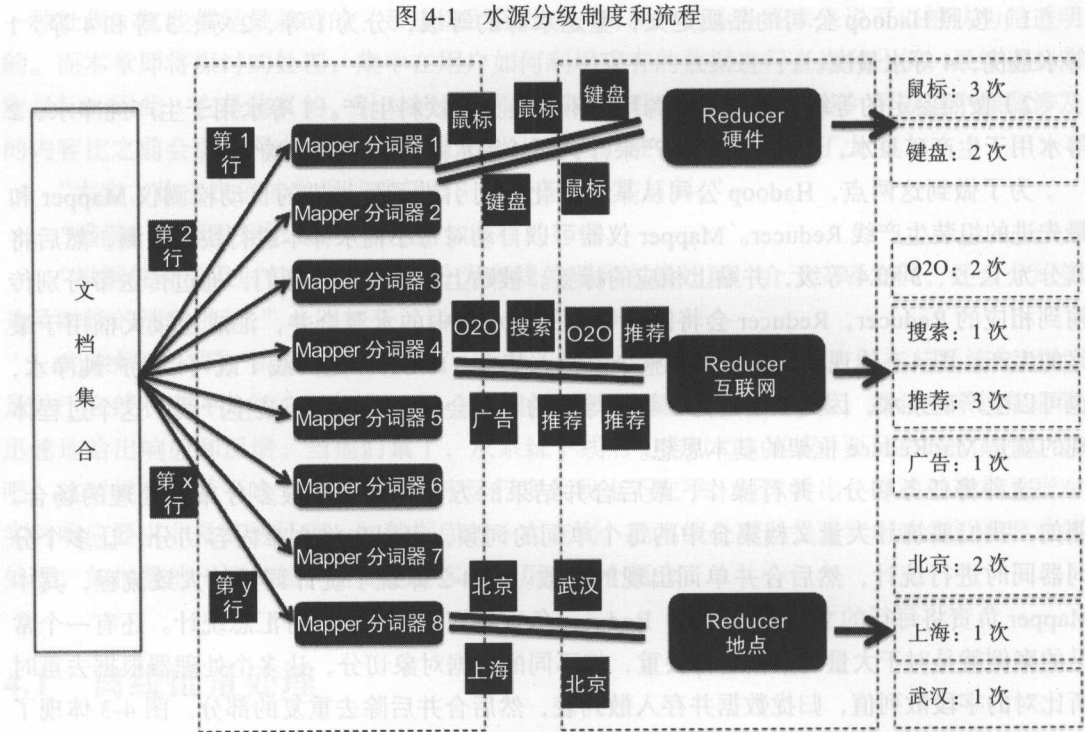
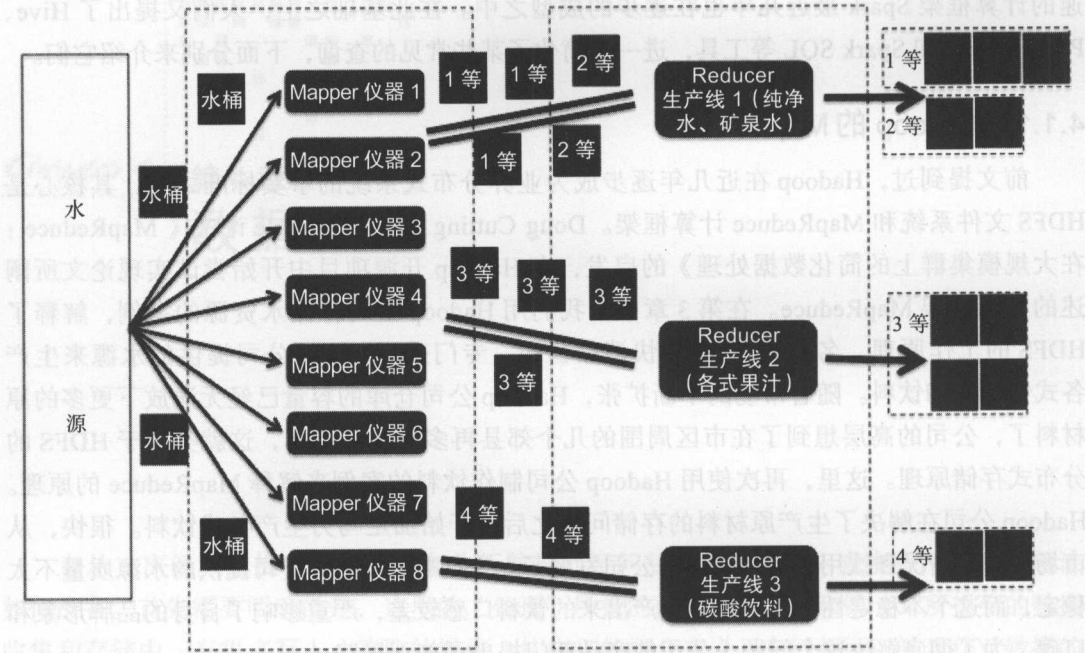


图 4-2 通过 MapReduce 框架统计词频

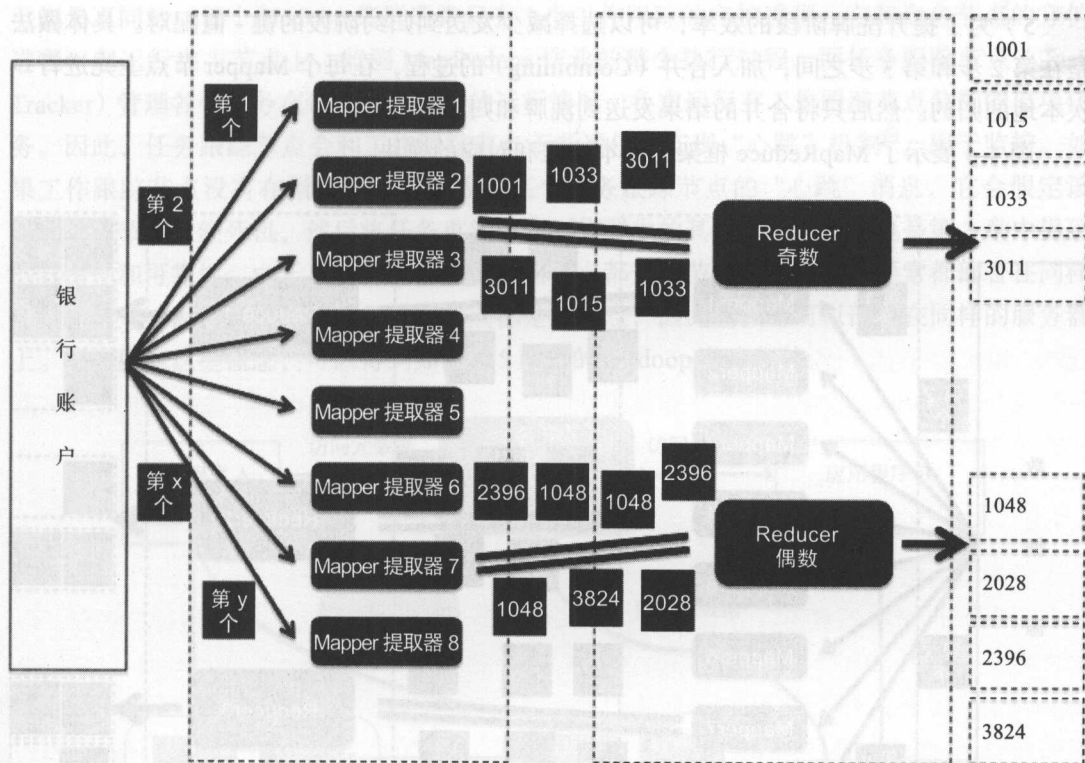


图 4-3 通过 MapReduce 框架对重复的记录去重

从这几个案例可以看出，这个框架的核心是散列表的映射结构，该结构在第 3 章的缓存设计中已经有所介绍。这再次证明散列是计算机领域中应用最为广泛的数据结构之一，对于 MapReduce 框架的运用也起到了至关重要的作用。在散列的基础上，MapReduce 框架包含几个重要的组成模块：

1) 数据分割 (Data Splitting)：将数据源进行切分，并将分片发送到 Mapper 上。例如对文档的每一行作为最小的处理单元。

2) 映射 (Mapping)：Mapper 根据应用的需求，将内容按照键 - 值的匹配，存储到散列结构  $\langle k1, v1 \rangle$  中。例如，将每行进行单词分词，然后生成  $\langle \text{北京}, 1 \rangle$  这样的配对，表示“北京”这个词出现了一次。

3) 洗牌 (Shuffling)：将键 - 值的配对不断地发给 Reducer 进行归约。如果存在多个 Reducer，还会使用分配 (Partitioning) 对 Reducer 进行选择。例如，“北京”、“上海”、“武汉”这种属于地名的单词，专门交给负责统计地点的 Reducer 来完成。

4) 归约 (Reducing)：分析接收到的一组键值配对，如果是键内容相同的配对，就将它们的值合并。例如，一共收到 12 个  $\langle \text{北京}, 1 \rangle$ ， $\{ \langle \text{北京}, 1 \rangle, \langle \text{北京}, 1 \rangle, \dots, \langle \text{北京}, 1 \rangle \}$ ，那么就将其合并为  $\langle \text{北京}, 12 \rangle$ 。最终“北京”这个单词的词频就统计为 12。



5) 为了提升洗牌阶段的效率, 可以选择减少发送到归约阶段的键 - 值配对。具体做法是在第 2 步和第 3 步之间, 加入合并 (Combining) 的过程, 在每个 Mapper 节点上先进行一次本地的归约。然后只将合并的结果发送到洗牌和归约阶段。

图 4-4 展示了 MapReduce 框架的基本流程和对应的模块。

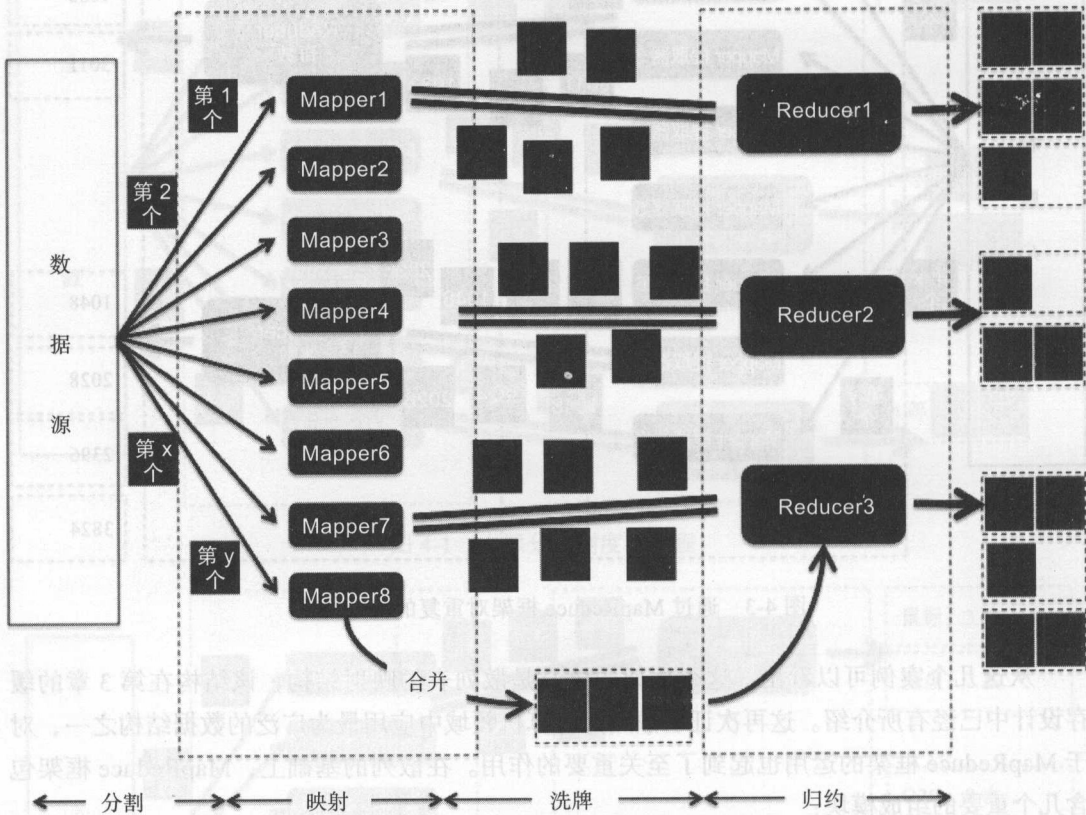


图 4-4 MapReduce 的基本框架

第 3 章介绍了 HDFS 中的 3 个概念: 命名节点 (Name Node)、次要命名节点 (Secondary Name Node) 和数据节点 (Data Node)。命名节点维护着系统的大量元数据, 负责管理文件系统的命名空间和控制外部的访问。次要命名节点和命名节点通信, 根据配置定期地获取其上的元数据快照, 达到备份和灾备的目的。而那些存储数据的节点则称为数据节点。理解了 MapReduce 的原理之后, 就可以将 Hadoop 的系统架构补充完整了, 这里增加了另外两个主要概念: 工作跟踪节点 (Job Tracker) 和任务跟踪节点 (Task Tracker)。工作跟踪节点是应用程序和 Hadoop 之间的纽带。一旦将代码提交到 Hadoop 集群上, 工作跟踪节点就会执行计划, 包括决定处理哪些文件、为不同的任务分配节点, 以及监控所有任务的运行。如果任务失败, 工作跟踪节点将根据配置打开重试机制, 自动重启任务。不过所分配的节点也

可能是不同的。每个 Hadoop 集群通常只有 1 个工作跟踪的守护进程，它和命名节点的守护进程一起运行在主节点上，监测 MapReduce 作业的整个执行过程。而任务跟踪节点（Task Tracker）管理各个任务在每个从节点上的运行情况，负责运行有工作跟踪节点分配的单项任务。因此，任务跟踪节点会和工作跟踪节点不断通信，实现“心跳”机制<sup>①</sup>，用于监控。如果工作跟踪节点没有在指定的时间内收到某个任务跟踪节点的“心跳”消息，它会假定该任务跟踪节点已经死机，然后将任务重新分配到集群中的其他节点上。这也是第 3 章中提到的容错性和可靠性。由于工作跟踪节点和命名节点都是主节点，因此它们通常都部署在同样的服务器上。而任务跟踪节点和数据节点都是从节点，因此它们也可以部署在同样的服务器上。综合所有这些概念，可以得到如图 4-5 所示的 Hadoop 基本架构。

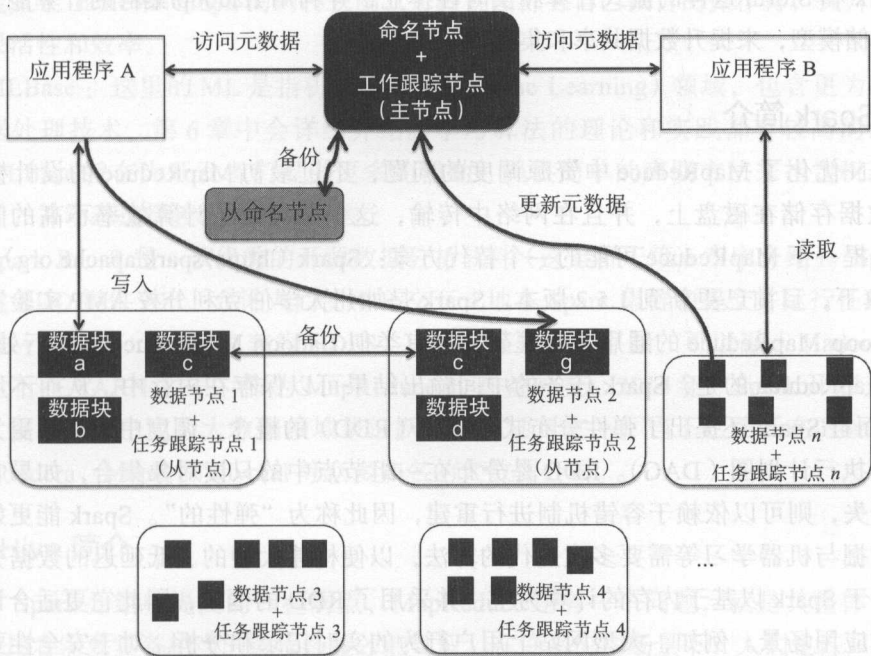


图 4-5 Hadoop 基本系统架构，包括 HDFS 和 MapReduce 框架

有了分布式文件系统 HDFS 和分布式计算框架 MapReduce 这两驾马车保驾护航，Hadoop 系统近几年的发展可谓风生水起。不过，人们也意识到 MapReduce 框架的一些问题。比如，拿工作跟踪节点 Job Tracker 来说，它是 MapReduce 的集中点，完成了太多的任务，造成了过多的资源消耗，存在单点故障的可能。而在任务跟踪节点 Task Tracker 端，用任务的数量来衡量负载的方式过于简单，没有考虑中央运算器 CPU、内存和硬盘等的使用

① 客户端定时发送简单的信息给服务器端表明它还在运作，每次发送的消息被形象地比喻为心电图中的“心跳”。如果服务器长时间无法收到这样的“心跳”，就会认为客户端已经宕机。

情况，有可能出现负载不均和某些节点的过载。Map 和 Reduce 任务的严格划分，也可能导致某些场合下系统资源没有被充分地利用。

面对种种问题，研发人员开始思考新的模式，包括 Apache Hadoop YARN (Yet Another Resource Negotiator)<sup>⑨</sup>等。Apache Hadoop YARN 是一种新的资源管理器，为上层应用提供了统一的 Hadoop 资源管理和调度。YARN 将工作跟踪节点 Job Tracker 的两个主要功能分成了两个独立的服务程序：全局的资源管理器 (Resource Manager) 和针对每个应用的主节点 (Application Master)。如此的设计是为了让子任务的监测进行分布式处理，大幅减少了工作跟踪节点 Job Tracker 的资源消耗。同时，这里说的应用既可以是传统意义上的 MapReduce 任务，也可以是基于有向无环图 (DAG) 的任务。因此，在 YARN 的基础上，甚至还可以运行 Spark 和 Storm 这样的流式计算和实时性作业，并利用 Hadoop 集群的计算能力和丰富的数据存储模型，来提升数据共享和集群的利用率。

### 4.1.2 Spark 简介

YARN 优化了 MapReduce 中资源调度的问题，不过最初 MapReduce 的设计模式要求将中间数据存储于磁盘上，并且在网络中传输，这仍然导致了计算效率不高的问题。因此，需要提一下 MapReduce 可能的一个替代方案：Spark (<http://spark.apache.org>)，它也在 Apache 旗下，目前已更新到 1.5.2 版本。Spark 是加州大学伯克利分校 AMP 实验室所开源的类 Hadoop MapReduce 的通用并行框架，拥有类似 Hadoop MapReduce 的并行处理模式。不同于 MapReduce 的是，Spark 任务的中间输出结果可以保存在内存中，从而不用再读写 HDFS。而且 Spark 还提出了弹性分布式数据集 (RDD) 的概念，调度中采用了更为通用的有向任务执行计划图 (DAG)。RDD 是分布在一组节点中的只读对象集合，如果它们中的一部分丢失，则可以依赖于容错机制进行重建，因此称为“弹性的”。Spark 能更好地适用于数据挖掘与机器学习等需要多次迭代的算法，以便构建大型的、低延迟的数据分析应用程序。由于 Spark 以基于内存的计算为主，并采用了 RDD 的框架，因此它更适合计算实时性更高的应用场景。例如，大型网站上用户行为的实时记录和分析。对于安全性要求很高的网站甚至可以用于识别一些非法的用户操作。此外，Hadoop 是采用 Java 语言编写的，与此不同的是，Spark 是通过 Scala 语言来实现的，它将 Scala 用作其应用程序的框架。这样，Scala 既可以像操作本地集合对象一样轻松地操作分布式数据集，同时也保持了 Spark 内核的简洁性。

Spark 的生态除了其核心部分外，还包括 Shark、Spark SQL、Spark Streaming、Spark GraphX、MLBase 和 Spark R 等。

❑ Shark 和 Spark SQL：Shark 可以理解为 Spark 上的 Hive (稍后会介绍 Hive，这个工具用于简化基于 MapReduce 的查询)，在实现上是将 HiveQL 翻译成 Spark 上的 RDD

<sup>⑨</sup> <http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/index.html>



操作。但由于整合度的问题, Shark 目前已经暂停开发。而 Spark SQL 是 Spark 1.0 版本后新推出的基于 Catalyst 的交互式 SQL 技术, 它和 Shark 类似, 不过比 Shark 支持更多的查询表达式, 也支持 Hive 的多种数据格式。另外, Hive 也刚刚启动了项目开始在 Spark 上构建交互式查询。

❑ Spark Streaming: 它可针对实时数据流进行高吞吐量、高容错的流式处理, 可以对数据进行映射 (Map)、归约 (Reduce) 和连接 (Join) 等高级操作, 并将结果输出到文件系统或数据库。稍后在实时数据处理部分还会介绍到这点。

❑ Spark GraphX: 提供了 Spark 中用于图计算的 API。图论对于大数据的分析也有重要的地位, 例如 Google 著名的 PageRank 理论<sup>①</sup>。GraphX 扩展了 Spark RDD 的抽象, 定义了 Table 和 Graph 的两种视图, 每种视图都有自己独自的操作符, 提升了系统的灵活性和效率。

❑ MLBase: 这里的 ML 是指机器学习 (Machine Learning) 领域, 包含更为高阶的数据处理技术, 第 6 章中会详细介绍。学习算法的理论和实践都有较高的门槛, 而 MLBase 旨在为开发者们提供更多的辅助, 提供简单的声明方法来指定机器学习任务, 并动态选择较优的学习算法。

❑ Spark R: R 是一款优秀的开源数据分析软件 (这点将于第 6 章中介绍)。Spark R 通过 RDD 提供 API, 允许人们使用 R 交互式地在 Spark 集群中提交并运行任务。

值得一提的是, Spark 也兼容 HDFS、HBase 等分布式存储, 可以融入 Hadoop 的生态系统, 运行在 YARN 之中, 作为 MapReduce 的有益补充。2014 年, Spark 开源生态系统得到了大幅增长, 已成为大数据领域最活跃的开源项目之一, 当下已活跃在 Hortonworks、IBM、Cloudera、MapR 和 Pivotal 等众多知名的大数据公司。

### 4.1.3 Hive 简介

除了 Spark 介绍中提及的一些弱点, MapReduce 还有一个问题, 就是其整体专业性比较强, 使用者需要具备相当的编程经验才能驾驭。在许多应用场景下, 人们也许只是想利用 Hadoop 进行一个简单的查询。为了这种需求兴师动众地写上一大堆的 MapReduce 代码得不偿失。那么, 有没有可能像关系型数据库的 SQL 语言那样, 只提供简单的命令和脚本, 就能获得强大的数据查询和修改功能呢? 下面来介绍 Hadoop 家族中的另一个利器: Hive。

Hive (<http://hive.apache.org/>) 是 Hadoop 项目中的另一个子项目, 它是建立在 Hadoop 基础之上的数据仓库工具, 可以存储、查询和分析存储在 HDFS 中的大规模数据, 目前最新的版本是 1.2.1。在第 3 章中, 我们提到了经典的关系型数据库使用结构化查询语言 SQL (Structure Query Language) 来查找数据。为了实现数据的提取、转化和加载 (Extraction, Transform, Load), Hive 也定义了一种简单的类 SQL 查询语言, 称为 HiveQL (Hive SQL)。

① 第 5 章信息检索会介绍 Web 图和 PageRank。



对于熟悉 SQL 的用户而言，HiveQL 的入门和使用非常方便。Hive 会将 SQL 语句转换为 MapReduce 任务在后台运行，因此用户不必开发专门的 MapReduce 应用，也不用关心具体的转换逻辑，非常适合用于数据仓库的统计分析。同时，HiveQL 也允许熟悉 MapReduce 框架的用户，开发自定义的 Mapper 和 Reducer，来处理内建模块无法完成的复杂工作。

从架构上看，Hive 主要包括如下模块：用户端、解释器、元数据存储和分析数据存储。

❑ 用户端：主要包含命令行（CLI）、客户端（Client）和 Web 图形化界面（WebGUI）。

最常用的是 CLI，它启动的时候会同时启动一个 Hive 守护进程服务，使用者可以交互式地输入命令并得到相应的结果输出。Client 是 Hive 的客户端，用户通过它连接到 Hive 的服务器。Client 模式启动的时候，需要启动 Hive 服务器所在的节点，并进行相应的配置。WebGUI 工具允许用户通过浏览器访问 Hive，使用前要启动 HWI 组件（Hive Web Interface）。

❑ 解释器：主要包含执行编译器、优化器和执行器，它们完成 HiveQL 查询语句的词法分析、语法分析、编译、优化及计划的生成。生成的查询计划也会存储在 HDFS 中，并在随后通过 MapReduce 框架调用执行。这也体现了 Hive 的核心思想之一，就是尽量简化 MapReduce 开发的工作量，使得某些操作和查询的复杂逻辑对使用者完全透明。

❑ 元数据存储：Hive 中的元数据包括表的名字、表的列、表分区、表数据所在的目录、是否为外部表，等等。尽管 Hive 采用 NoSQL 的方式进行工作，但它仍然使用关系型数据库存储元数据，这点主要是考虑到元数据的规模较小，而对读写同步的要求很高。此外，将元数据的存储从 Hive 的数据服务中解耦出来，可以大大减少执行语义检查的时间，也能提高整个系统运行的健壮性。常用的关系型数据库配置是 MySQL 或 Derby 嵌入式数据库。

❑ 分析数据存储：Hive 用于分析的海量数据都存储在 HDFS 之中，支持不同的存储类型包括纯文本文件、HBase 等文件。一旦解释器接受了 HiveQL，那么 Hive 将直接读取 HDFS 的数据，并将查询逻辑转化成 MapReduce 计算来完成。

整个 Hive 加上 Hadoop 的大体架构如图 4-6 所示。

Hive 中所有的数据都存储在 HDFS 中，并没有专门的存储格式，也没有为数据建立索引，它可以非常自由地组织表结构。使用者只需要在创建表的时候指定字段列分隔符和记录行分隔符，Hive 就可以进行解析。尽管如此，Hive 的数据模型仍然包括几个主要概念：数据库（Database）、表（Table）、分区（Partition）和桶（Bucket）。

❑ 数据库：它的作用是将用户的应用隔离到不同的数据模式中，Hive 0.6.0 之后的版本都支持数据库，相当于关系型数据库里的命名空间（Namespace）。

❑ 表：Hive 的表和数据库中的表在概念上非常接近，在逻辑上，其由描述表格形式的元数据和存储于其中的具体数据共同组成，可以分为托管表和外部表。托管表在 Hive 中有一个对应的目录，所有的数据都存储在这个目录中。而外部表的数据文件

可以存放在 Hive 仓库以外的分布式文件系统上。表删除的 DROP 命令对于这两种类型产生的效果也不同，对托管表执行 DROP 命令的时候，会同时删除元数据和其中存储的数据，而对外部表执行该命令的时候，则只能删除元数据，而不会删除外部分布式系统上所存储的数据。

□ 分区：Hive 中的分区方式和数据库中的差异很大，它的概念是根据分区列对表中的数据进行大致地划分。这里，分区列不是表里的某个字段，而是一个独立的列。前面提到过 Hive 表就是通过分布式文件系统的目录来实现的，那么相应地，表的分区在 Hive 存储上就体现为主目录下的多个子目录，而子目录的名称就是分区列的名称。使用分区的好处在于，查询某个具体分区列里的数据时不用进行全表扫描，可以大大加快范围内的查询。

□ 桶：表和分区都是在目录级别上进行数据的拆分，而桶则是对数据源数据文件本身进行数据拆分。使用桶的表会将源数据文件按一定的规律拆分成多个文件。

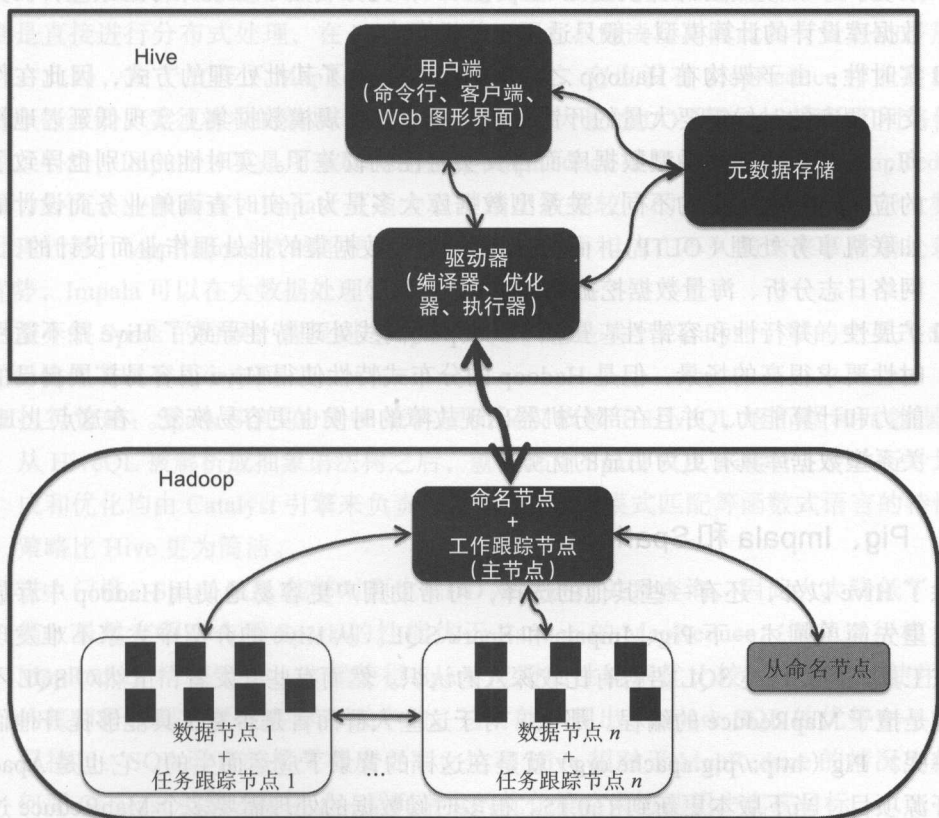


图 4-6 Hive 的基本系统架构、底层存储和计算都通过 Hadoop 来实现

“小明哥，之前你介绍完 Hadoop，我感觉 MapReduce 上手还有些困难。目前看来，

Hive 能省不少事啊，如果理解了 SQL，那么不懂 MapReduce 也没关系，照样能完成数据的查询和修改啊。这样使用 HDFS 中的数据岂不是和关系型数据库非常类似了？”

“确实如此，因此 Hive 经常用于大数据的报表生成。比如，对于存储在 HDFS 和 HBase 中的用户行为数据，可以通过 HiveQL 查询每日的用户访问量 UV（Unique Visitor）、页面浏览量 PV（Page View）、订单转化率，等等。不过我们也要注意，Hive 与关系型的 SQL 数据库毕竟还是有所不同，应用场景也有差异。下面会简要地总结一下。”

- ❑ 查询语言：HiveQL 与大部分的 SQL 语法兼容，但并不是完全地支持 SQL 标准。由于底层依赖于 Hadoop 的平台，因此 HiveQL 不支持更新操作，也不支持索引和事务，子查询和连接（Join）操作也很有限。另外，HiveQL 还有些特点是关系型 SQL 所无法企及的，比如和 MapReduce 计算过程的集成和多表查询。
- ❑ 存储方式和计算模型：Hive 和关系型数据库相比，存储和计算的方式也不同。Hive 使用的是 Hadoop 的 HDFS 分布式文件系统，而关系型数据库则是服务器本地的文件系统。Hive 使用的计算模型是 MapReduce，充分利用了多机并行的原理，而关系型数据库设计的计算模型一般只适用于单机模式。
- ❑ 实时性：由于架构在 Hadoop 之上，Hive 也继承了其批处理的方式，因此在作业提交和调度的时候需要大量的开销，并且不能在大规模数据集上实现低延迟地快速查询，自然相较于关系型数据库而言其实时性就较差了。实时性的区别也导致了两者的应用场景有很大的不同，关系型数据库大多是为了实时查询的业务而设计的，例如联机事务处理（OLTP）。而 Hive 则是为大数据集的批处理作业而设计的，例如，网络日志分析、海量数据挖掘等。
- ❑ 扩展性、并行性和容错性：虽然 Hadoop 的离线处理特性导致了 Hive 并不适用于实时性要求很高的场景，但是 Hadoop 的分布式特性使得 Hive 很容易扩展自己的存储能力和计算能力，并且在部分机器出现故障的时候也更容易恢复。在这点上 Hive 比关系型数据库具有更为明显的优势。

#### 4.1.4 Pig、Impala 和 Spark SQL

除了 Hive 以外，还有一些其他的选择，可帮助用户更容易地使用 Hadoop 中存放的数据，这里先简单阐述一下 Pig、Impala 和 Spark SQL。从 Hive 的介绍中大家不难发现，要使用该工具，需要对类 SQL 语言有比较深入的认识。然而有些开发者，虽然对 SQL 不甚理解，但是擅于 MapReduce 的编程。那么，对于这些人群而言是否有工具能够提升他们的生产效率呢？Pig（<http://pig.apache.org>）就是在这样的背景下应运而生的，它也是 Apache 旗下的开源项目，当下版本更新到了 0.15。很多时候数据的处理需要多个 MapReduce 过程才能实现，数据处理过程与可能的数据转换也可能很困难。而 Pig 为大型数据集的处理提供了更高层次的抽象，以及更丰富的数据结构。从抽象层次来看，它提供了脚本语言 Pig Latin，该语言的编译器会将数据分析请求转换成一系列经过优化处理的 MapReduce 运算，可以认



为是 SQL 的一个面向过程的简化版本。其中，一条语句就是一个操作，与数据库的表类似。同时，Pig 还拥有大量的数据类型，不仅支持包、元组和映射等高级概念，还支持简单的数据类型。Pig 的比较运算符也相对完整，包括使用正则表达式的丰富匹配模式。因此，有了 Pig，用户不一定需要懂得 SQL 的语法和含义也能控制 MapReduce 的作业，同时又能简化 MapReduce 的开发和不同的数据之间的转换。

另一个执行于现有 Hadoop 基础设施上的互动 SQL 查询引擎是 Impala<sup>①</sup>，它是 Cloudera 公司主导开发的查询系统，目前的最新版本是 2.1。类似 Apache Hive，Impala 也能通过类 SQL 的语言查询存储在 HDFS 和 HBase 中的 PB 级大数据。不过，Impala 考虑了实时性更强的需求，在设计上和 Hive 有所不同。Hive 采用的方法是将 SQL 查询转化成 MapReduce 任务，这仍然是一个批处理过程，故而难以满足查询的交互性。相比之下，Impala 的速度之快就成了它的一大特色。为了实现这一点，Impala 参考了 Google 的交互式数据分析系统 Dremel。Impala 使用 Parquet 实现了列存储，并借鉴了 MPP 并行数据库的思想。同时，它采用 HiveQL 和 JDBC 等接口，进行全局统一的元数据存储和读取。对于用户查询则是直接进行分布式处理，在 HDFS 或 HBase 上本地读写，因此具有良好的扩展性和容错性。此外，由于放弃了 MapReduce 的运行框架，它也没有 MapReduce 作业启动、洗牌、排序等开销，无须将中间结果写入磁盘，节省了大量的 I/O 开销，也降低了网络传输的数据量。当然，Impala 并不是用来取代现有的 MapReduce 框架的，而是作为 MapReduce 的一个强力补充。一般而言，Impala 更适合处理输出数据较小的查询请求，而对于大数据量的批处理任务，MapReduce 依然是更好的选择。有理由相信在不久的将来，借助处理速度上的优势，Impala 可以在大数据处理领域占得一席之地。

之前介绍 Spark 的时候，曾提到过 Spark SQL，它是基于 Catalyst 引擎的交互式 SQL 技术，主要优化了以下几个方面。

- ❑ 执行策略：Spark SQL 在 Hive 兼容层面仅仅依赖于 HiveQL 解析器和元数据存储。从 HiveQL 被解析成抽象语法树之后，就全部由 Spark SQL 来接管了。执行计划的生成和优化均由 Catalyst 引擎来负责，借助 Scala 的模式匹配等函数式语言的特性，其策略比 Hive 更为简洁。
- ❑ 进入门槛：Spark SQL 能够对原生 RDD 对象进行关系查询，因此大大降低了用户门槛。虽然在很多方面 Spark 的性能优于 Hadoop 的 MapReduce，但其运行模型也比 MapReduce 精细不少，这就使得 Spark 应用的性能调优比较复杂。单纯使用 Spark 的接口开发是需要花些学习成本的。这时就体现出了 Spark SQL 的优势——相比底层接口，SQL 语言的接受程度更高，这和 Hive 相对于 MapReduce 的情况类似。更何况 Catalyst 引擎还提供了一系列常见的优化策略来协助用户实现目标。
- ❑ 对 Hive 的依赖：相对于 Shark，Spark SQL 进一步削减了对 Hive 的依赖，不再需要

① <http://www.cloudera.com/content/www/en-us/products/apache-hadoop/impala.html>



自行维护打了补丁的 Hive 分支。因此，Shark 后续将全面采用 Spark SQL 作为引擎，而不仅仅是查询优化方面。

## 4.2 提升及时性：消息机制

“小明哥，我理解 Hadoop 的 HDFS 和 MapReduce 使得大规模的数据并行处理成为可能。不过，并行处理的优势还是体现在批量的离线处理上。当新数据源源不断地产生，或者原有的数据非常频繁地更新时，我们又该如何处理才能快速地体现变化的结果？”

“这正是我想阐述的。不过，在此之前，首先要介绍一种消息的机制，可以在一定程度上改善及时性的问题。还是从生产饮料的案例说起吧。”

自从使用了 Mapper 水质鉴定器和 Reducer 的组合流水线，Hadoop 公司的生产业务开展得非常顺利。当生产任务繁忙的时候，这样的自动化流程保持了很高的效率。不过，到了销售的淡季，市场的需求骤降，每次的订单量十分有限，而且呈现零散分布的状态。公司发现在这种情况下，每次批量生产的数量太少，很多时候生产线在空转，导致效率大幅降低，平均成本明显上升。于是，公司就将单生产的模式改为定期生产，不再是来了订单就生产，而是每周生产一次。但是，新的问题又产生了：一周的间隔时间有点长，存货往往又不够，很容易导致市面上经常性地缺货。这该如何是好呢？公司的高层又犯难了。有人提出这样的建议：当接收到一批新的生产订单时，公司就将订单记录下来。当新订单累积到足够的量时就开动生产，如果没有则继续等待下一批新订单。这种方案的好处在于，公司既没有必要一直开动生产线浪费资源，也没有必要积压很长的周期，完全是根据订单的需求波动来制定生产的节奏，最后被公司采纳并命名为“按需生产”。

回到正题，我们可以认为 MapReduce 计算就好比流水生产线的批量作业，而新的数据或数据变更则可以被认为新的订购需求。如果能让数据变化所导致的计算结果能够及时体现出来，一种做法就是不停地进行 MapReduce 运算。但是，如果实际并没有足够的数据更新，那么很多时候 MapReduce 的运算都是浪费的，计算的结果并没有变化，这就好比流水线空转。另一种做法就是定时地进行 MapReduce 运算，不过这样可能会造成变更的延时。如果采纳“按需生产”的建议，根据变化的规模来定制 MapReduce 的计算频率，则相对更为合理一些。实质上，我们可以认为该方案就是消息机制的核心思想：当信息源发生变动的时候，将其通知给需要感知这个变动的监控者，然后监控者根据应用场景来决定如何处理。有了核心思想，我们再来看一下几个主要的概念：消息的生产者、消息的消费者、消息队列，以及传递模式。

消息的生产者和消费者很容易理解，分别是消息的发送方和接收方。有了消息的机制以后，使用者还可以利用消息实现消息队列，对处理请求进行缓冲。队列的好处在第 2 章中也有所提及。如果生产源主动推送的数据不能及时被采集端处理，将会带来很多复杂的后续问题。因此推送的模式中常常需要加上队列。经过这样的异步处理，队列的缓冲可以对处理

端起到保护作用，保证其不容易被数据洪流冲垮。队列最为常见的一种实现就是消息的队列，其优势在于：

- 采用了异步通信的模式。消费者不必在接到消息后立即对生产者的请求进行处理，同样生产者可以在发送消息后进行其他的工作，不用等待消费者的回应。
- 生产者和消费者的松耦合。两个角色不必都正常运行，如果服务器崩溃或网络故障导致消息无法送达，接收端不会受到这种异常的干扰，如果消息队列支持持久化，那么还能确保消息不会丢失。

对于消息传递的方式，有两种基本的模型：点对点模型 P2P（Point to Point）和发布 / 订阅模型（Publish / Subscribe）。

点对点的模型用于消息生产者和消息消费者之间点到点的通信。生产者将消息发送给由某个名字或 ID 唯一指定的特定消费者。这个唯一标识可以对应于消息服务中的一个队列，在消息被传送到消费者之前它会被存储在这个队列中。队列可以持久化，以保证消息服务在故障恢复后仍然能够传递消息。图 4-7 展示了点对点的模型。

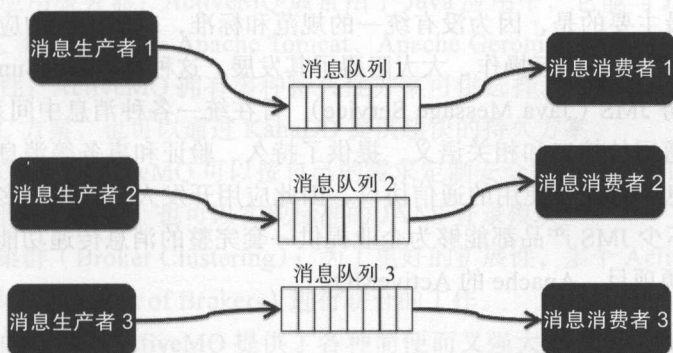


图 4-7 点对点的消息传送模式

发布 - 订阅模型用名为主题（Topic）的内容分层结构代替了点对点模型中的唯一目的地。生产者将自己的消息发布到分层结构中的一个主题。而希望接收这些消息的消费者订阅了这个主题。如此一来，订阅者可以接收该主题和其子主题发表的所有消息。图 4-8 展示了发布和订阅的模型。从图 4-8 中可以看出，多个应用程序可以就一个主题发布和订阅消息，而应用程序对其他人是匿名的。其中的关键在于代理（Broker）起着中介的作用，将一个主题已发表的消息传送给该主题的所有订阅者。

无论是哪种消息传送模式，都可以提升数据更新的及时性，并对复杂的系统架构进行解耦。举个例子，对于重点观察的用户行为，如果还是通过 Flume 这样的批量采集方式，可能无法达到业务方提出的实时监控和分析的需求。而消息的机制可以很好地解决这个问题。另外，由于消息机制的实时性更强，通常它还会和稍后介绍的 Spark Streaming、Storm 这样的流式计算结合起来使用。

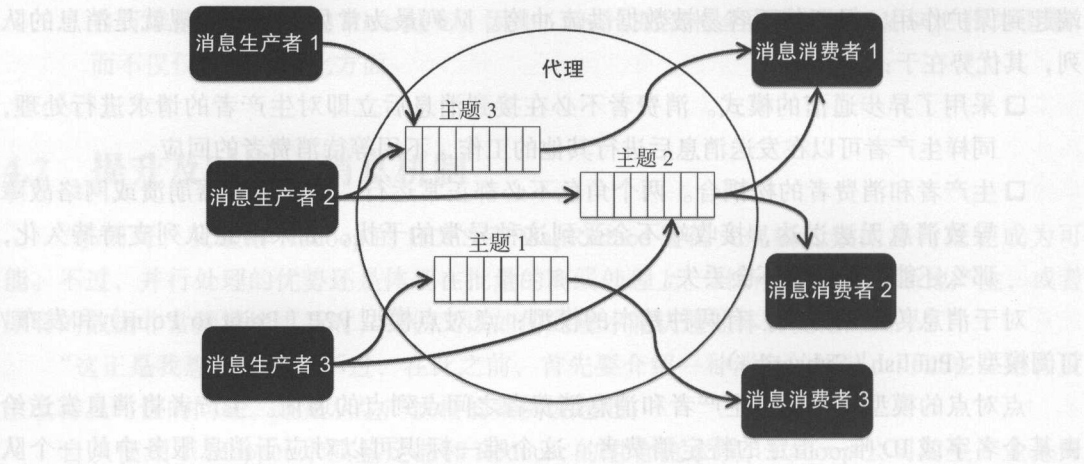


图 4-8 发布 / 订阅的消息传送模式

从 20 世纪 90 年代初开始，消息机制相关的技术得到了长足的发展。同时，人们也发现了不少问题。最主要的是，因为没有统一的规范和标准，基于消息的应用通常都不可移植，不同的消息机制也不能互操作，大大阻碍了其发展。这种形势下，Sun 及其伙伴公司提出了 Java 消息服务 JMS（Java Message Service），旨在统一各种消息中间系统的接口规范。JMS 定义了一套通用的接口和相关语义，提供了持久、验证和事务等消息服务。该项规范并没有指定在消息节点间所使用的通信协议，因此应用开发人员不用过多地考虑底层的细节。时至今日，不少 JMS 产品都能够为企业提供一套完整的消息传递功能。这里将介绍一款常见的 JMS 开源项目，Apache 的 ActiveMQ。

### 4.2.1 ActiveMQ 简介

ActiveMQ（<http://activemq.apache.org>）是 Apache 出品的，非常流行的一款开源消息系统，当下最新的版本是 5.13。21 世纪初，一群热爱技术的开发者在一起搭建了开放源码的 J2EE 服务器：Apache Geronimo。很快，他们发现 Geronimo 需要一款基于 JMS 来实现的消息代理机制。当时市面上大多数的消息中间件都是商业版，使用成本比较昂贵，而且不利于二次开发和维护。很显然，市场急需一个开源的系统，而这就最终促使了 ActiveMQ 的诞生。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS 实现，它为应用的消息传递提供了高可用、可扩展的技术方案，也在一定程度上保证了出色的性能和安全性。

ActiveMQ 的主要目标是在尽可能多的跨平台和跨语言上提供一个统一的、标准的消息驱动的应用集成，因此它提供了许多特性。

□ 实现松耦合：如前面所述，消息驱动的应用可以在实现松耦合的同时，及时地感知变化的存在。ActiveMQ 同样以异步的形式提供松耦合的应用架构，应用对 ActiveMQ



的调用不依赖于其他任何应用，消息的生产者并不关心消息是如何及何时被传递的，同样消息的消费者也不关心消息从哪里来或如何到来。

□ 遵循 JMS 规范：ActiveMQ 的各种特性是 JMS1.1 规范的实现。JMS 规范保证了同步和异步消息传递、一次和仅一次的传递、对于订阅者的消息持久化，等等。基于 JMS 规范使得 ActiveMQ 和其他消息提供者拥有类似的基本特性。

□ 支持多种连接协议：ActiveMQ 提供了各种连接选择，包括 HTTP、HTTPS、SSL、TCP、UDP、XMPP、IP 多点传送等。由于很多既有系统使用的协议无法被轻易改变，大量的连接协议支持使 ActiveMQ 具有更好的灵活性，大幅降低了使用的门槛和成本。

□ 提供多种客户端接口：虽然 ActiveMQ 代理（Broker）是运行在 Java 虚拟机上的，但 ActiveMQ 的客户端却对多种编程语言提供了 API 接口，例如 C/C++、.NET、PHP、Python 和 Perl 等版本。这使得我们在 Java 之外的其他语言中，也能够使用 ActiveMQ 的多种特性。

□ 支持多种应用服务器：ActiveMQ 最常用于 Java 应用中，它能与 Java 应用服务器很好地集成。例如，Jetty、Apache Tomcat、Apache Geronimo 和 JBoss 等。

□ 提供持久性：ActiveMQ 拥有多种持久性方案可供选择。例如，ActiveMQ 既支持标准的 JDBC 方案，也可以通过 KahaDB 提供超快的持久方案。

□ 提供安全保障：ActiveMQ 可以按自己的需求定制安全等级，既可以通过配置文件提供简单的验证和授权，也可以实现标准的 JAAS 登录模块。

□ 支持代理集群（Broker Clustering）：为了更好的扩展性，多个 ActiveMQ 代理可以通过代理网络（Network of Brokers）进行联合的工作。

□ 提供简单的管理：ActiveMQ 提供了各种简便而又强大的管理方式，除了 Java 语言中最基本的 JConsole，还有 ActiveMQ Web Console、消息报告和各种系统日志等。

并非所有的消息机制都一定要通过 JMS 来实现，例如 Apache Kafka。下面简单介绍这款优秀的消息中间件。

## 4.2.2 Kafka 简介

Apache Kafka (<http://kafka.apache.org>) 是 LinkedIn 公司设计和开发的高吞吐量的分布式发布订阅消息系统，其内在设计就是分布式的，具有良好的可扩展性，目前的版本更新到了 0.9。Kafka 的创造者们在使用之前的一些消息中间件时，发现如果严格遵循 JMS 的规范，虽然消息投递的成功率非常之高，但是会增加不少额外的消耗，例如 JMS 所需的沉重消息头，以及维护各种索引结构的开销等。最终导致系统性能很难有进一步的突破，不太适合海量数据的应用。因此，他们并没有完全按照 JMS 的规范来设计 Kafka，而是对一些原有的定义做了简化，大幅提升了处理性能，同时对传送成功率也有一定的保证。总体看来，Kafka 有如下特性。



□ 高性能存储：通过特别设计的磁盘数据结构，保证时间复杂度为  $O(1)$ <sup>①</sup> 的消息持久化，这样数以 TB 的消息存储也能够保持良好的稳定性能。此外，被保存的消息可以多次被消费，用于商务智能 ETL 和其他一些实时应用程序。

□ 天生分布式：Kafka 被设计为一个分布式系统，它利用 ZooKeeper 来管理多个代理 (Broker)，支持负载均衡和副本机制，易于横向地扩展。ZooKeeper 旨在构建可靠的、分布式的数据结构，这里用于管理和协调 Kafka 代理。当系统中新增了代理，或者某个代理故障失效时，ZooKeeper 服务会通知生产者 and 消费者，让它们据此开始与其他代理协调工作。

□ 高吞吐量<sup>②</sup>：由于存储性能的大幅提升，以及良好的横向扩展性，因此即使是非常普通的硬件 Kafka 也可以支持每秒数十万的消息流，同时为发布和订阅提供惊人的吞吐量。

□ 无状态代理：与其他消息系统不同，Kafka 代理是无状态的。代理不会记录消息被消费的状态，而是需要消费者各自维护。

□ 主题 (Topic) 和分区 (Partition)：支持通过 Kafka 服务器和消费机集群来分区消息。一个主题可以认为是一类消息，而每个主题可以分成多个分区。通过分区，可以将数据分散到多个服务器上，避免达到单机瓶颈。更多的分区意味着可以容纳更多的消费者，有效提升并发消费的能力。基于副本方案，还能够对多个分区进行备份和调度。

□ 消费者分组 (Consumer Group)：Kafka 中每个消费者均属于一个分组，每个分组中可以有多个消费者。主题中的某条消息可以被多个分组获得，不过同一分组中，只有一个消费者会获得该消息。

图 4-9 是 Kafka 系统的大体架构。当然，性能的显著提升并不一定意味着传递可靠性的下降。对于 JMS 规范的实现而言，消息传输的方式非常直接：有且只有一次。而在 Kafka 中则有所不同，可以分为 3 个档次。

□ 最多一次 (At Most Once)：这个与 JMS 中的非持久化消息类似，只发送一次，无论成败都不会重发。如果在消息处理过程中出现了异常，导致部分消息未能继续处理，那么此后未处理的消息将不能被获取。

□ 至少一次 (At Least Once)：消息至少发送一次，如果消息未能接受成功，可能会重发，直到接收成功为止。消费者获取并处理消息，但是如果发生异常导致状态保存操作未能执行成功，那么接下来再次获取时可能获得的是上次已经处理过的消息。

□ 正好一次 (Exactly Once)：消息只会发送一次，这其实和关系型数据中的事务概念一致。目前 Kafka 并没有严格地实现基于两阶段提交的事务机制。

考虑到重复接收数据总比丢失数据要好，通常情况下 Kafka 的“至少一次”机制是使用者的首选。整体而言，对于一些常规的消息系统，Kafka 是个理想的选择。内在的分布式设

① 时间复杂度的概念会在第 7 章性能部分具体介绍，这里可以简单地理解为时间的开销。

② 吞吐量同样会在第 7 章性能部分具体介绍，这里可以简单地理解为单位时间内处理的数量。

计、分区和副本，使得其具有良好的扩展性、容错性和性能优势。不过，目前 Kafka 并没有提供 JMS 中的事务性消息传输，无法严格地保证消息一定被处理，或者只被处理一次，适合那些对一致性要求不高的应用场景。

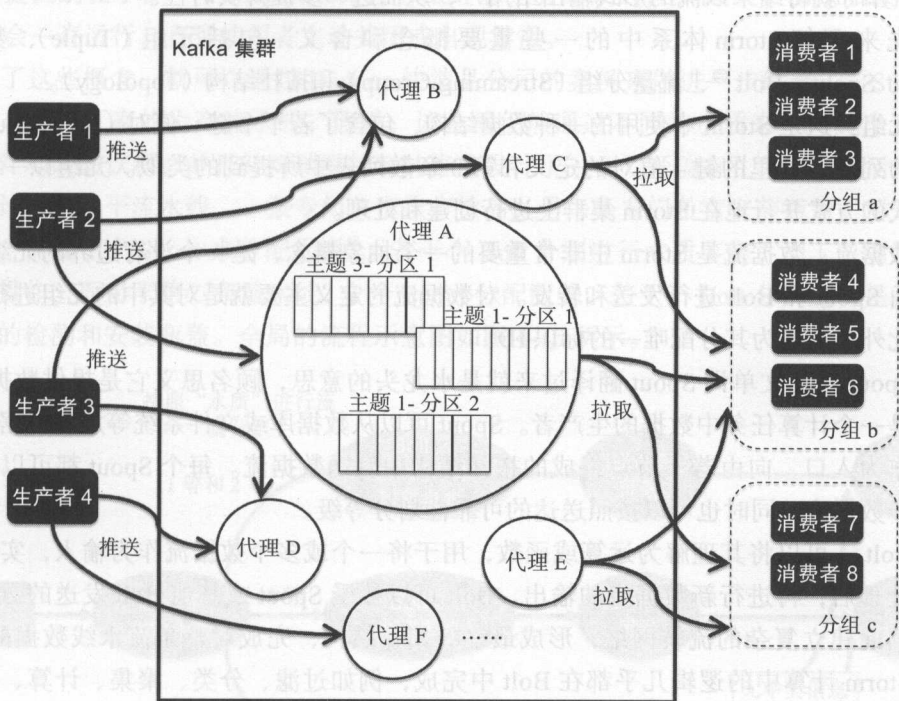


图 4-9 Kafka 的整体架构

## 4.3 在线实时处理

“那么，小明哥，我的理解是，消息机制可以及时地通知应用接入方，何时、哪里发生了变化，然后应用方就可以进行相应的处理。比起全量的批处理，这种增量的方式可以提升实时性。不过，是否有可能在数据变化发生的同时，就直接将处理操作完成呢？这样岂不是更加及时？”

“大宝，你不愧是做 IT 出身的，思路很对，确实，人们为在线处理设计了更为实时的方案以响应快速的数据变化，例如流式计算。下面介绍其中两个比较常见的开源项目——Storm 和 Spark Streaming。”

### 4.3.1 Storm 简介

Storm (<http://storm.apache.org>) 源于 Twitter 公司，目前已经成为 Apache 顶级的开源项

目，最新的版本更新到了 0.10.0。Storm 是一个分布式的、容错的实时计算系统，除了用于实时分析外，它还可用于在线数据挖掘、机器学习<sup>①</sup>、商务智能 ETL、分布式远程调用等领域。Storm 为分布式实时计算提供了一组通用原语，这是管理队列及工作者集群的另一种方式。在计算时就将结果以流的形式输出给用户，从而进一步提升实时性。

首先来理解 Storm 体系中的一些重要概念和含义，包括元组 (Tuple)、数据流 (Stream)、Spout、Bolt<sup>②</sup>、流量分组 (Streaming Group) 和拓扑结构 (Topology)。

- ❑ 元组：这是 Storm 中使用的一种数据结构，包含了若干个键-值对 (Key-Value Pair) 的列表，这里的键-值对的定义和第 3 章散列表中所提到的类似。元组以一种分布式的方式并行地在 Storm 集群上进行创建和处理。
- ❑ 数据流：数据流是 Storm 中非常重要的一个抽象概念，是一个没有边界的元组序列，由 Spout 和 Bolt 进行发送和转发。对数据流的定义主要就是对其中的元组进行定义，此外还需要为其分配唯一的标识 ID。
- ❑ Spout：英文单词 Spout 翻译过来就是水龙头的意思，顾名思义它是提供数据源的，是一个计算任务中数据的生产者。Spout 可以从数据库或文件系统等加载数据，然后作为入口，向由若干节点组成的拓扑结构中发送数据流。每个 Spout 都可以发送多个数据流，同时也可以按照送达的可靠性划分等级。
- ❑ Bolt：可以将其理解为运算或函数，用于将一个或多个数据流作为输入，实施加工处理后，再进行新数据流的输出。Bolt 可以接受 Spout 或其他 Bolt 发送的数据，并据此建立复杂的流转网络，形成最终的拓扑结构，完成对整条流水线数据的操作。Storm 计算中的逻辑几乎都在 Bolt 中完成，例如过滤、分类、聚集、计算、查询数据库等。
- ❑ 流量分组：它决定了 Spout 和 Bolt 节点之间相互连接的方式，主要分为以下几种类型。
  - 洗牌分组 (Shuffle Grouping)：随机地将元组分发到各个 Bolt 上，理论上这样做的结果是每个 Bolt 都会接收到同样数量的元组。
  - 按字段值分组 (Fields Grouping)：按照指定的元组字段来进行分组。例如，按照“水质”来划分，那么具有同等质量的水源会被分到一组，发送到同一个或同一组 Bolt 上。这个逻辑和 Hadoop 中的 MapReduce 框架非常相似，这样一来，数据流上游的 Spout 或 Bolt 节点就和 Mapper 比较接近，而下游的 Bolt 节点则和 Reducer 比较接近。
  - 广播 (All)：所有的元组都会发送到所有的 Bolt 上。
  - 全局 (Global)：所有的元组都发送到全局指定的某个 Bolt 上。
  - 不做指定 (None)：目前等同于洗牌分组，将来可能会进行新的定义扩充。
  - 指定分组 (Direct)：明确指定元组发送到哪个确切的 Bolt 上。

① 第 6 章将会介绍数据挖掘和机器学习。

② 不得不承认，这里 Spout 和 Bolt 很难找到既贴切又得体的翻译。



□ 拓扑结构：它是由流量分组连接起来的 Spout 和 Bolt 节点网络。在 Storm 中，一个实时计算应用程序的逻辑被封装在一个拓扑对象中，也可以称为计算拓扑。如果和 Hadoop 的生态系统对比，拓扑结构类似于 MapReduce 的任务，但是它们之间的关键区别在于，一个 MapReduce 任务最终总是会结束的，然而一个 Storm 的拓扑结构会一直运行，直到使用者主动关闭或出现异常。

有了这些概念，就可以通过 Hadoop 快消品公司的案例来做进一步解释了。Storm 的拓扑结构非常像厂家的生产流水线。假设，该公司仍然通过不同的水质来生产纯净水、矿泉水、果汁和碳酸饮料，并且针对碳酸饮料的推广设计了一个“开盖有礼”的抽奖活动。那么可以设计两条主干流水线，一条专门负责生产饮品，另一条专门负责安装瓶盖。对于第一条流水线，需要 1 个 Spout 来分配水源<sup>①</sup>，若干个 Bolt 分别进行水质鉴定、加工、灌装和安装瓶盖等操作。而对于第二条流水线，1 个 Spout 分配瓶盖，若干个 Bolt 分别进行是否包含抽奖信息的检测和安装瓶盖。全局的流程示意图如图 4-10 所示。

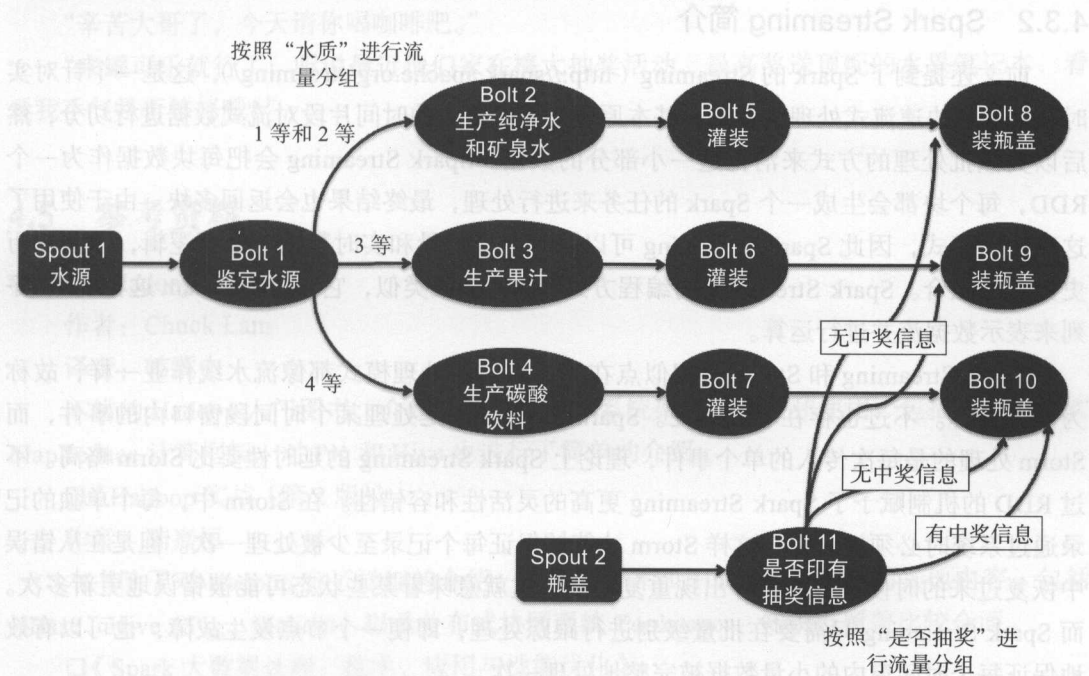


图 4-10 Storm 拓扑和生产流水线的类比

从架构的角度来理解，Storm 的集群主要包含两种节点：主节点 Nimbus 和工作节点 Supervisor，它们都是无状态的，可以从失败中快速恢复，健壮性较好。Nimbus 负责管理、协调和监控在集群上运行的拓扑结构，包括拓扑的发布、任务指派、出错后的恢复等。从这点上看，其功能和 Hadoop 集群中的工作跟踪节点（Job Tracker）非常像。Supervisor 在接

① 此处水源和 Spout 的水龙头含义没有直接关联，纯属巧合。



收到 Nimbus 分配的任务之后，会启动名为 Worker 的进程来完成工作。每个 Worker 负责一个拓扑结构，而一个 Supervisor 可以启动多个 Worker，并负责管理它们，类似 Hadoop 中任务跟踪节点（Task Tracker）的角色。此外，Storm 同样是利用 ZooKeeper 来管理节点的集群的，例如任务的分配情况、各个 Worker 的状态、Supervisor 之间的 Nimbus 的拓扑度量等。Nimbus 和 Supervisor 节点之间的通信也是结合 ZooKeeper 的状态变更通知和监控通知来处理的。

此外，Storm 的编程模型也类似于 MapReduce，并降低了实时处理的复杂性。系统的底层设计使用了消息队列，保证了数据能得到快速的处理。Storm 还可以提供可靠的处理等级，保证每条数据至少能得到一次完整的处理。任务失败时，它也会负责重试。同时，Storm 还支持各种编程语言。除了默认的选择，若要增加对其他语言的支持，只须实现一个简单的 Storm 通信协议即可。

### 4.3.2 Spark Streaming 简介

前文还提到了 Spark 的 Streaming (<http://spark.apache.org/streaming/>)，这是一个针对实时数据进行快速流式处理的系统，基本原理是按照很小的时间片段对流式数据进行切分，然后以类似批处理的方式来消化这一小部分的数据。Spark Streaming 会把每块数据作为一个 RDD，每个块都会生成一个 Spark 的任务来进行处理，最终结果也会返回多块。由于使用了这种设计模式，因此 Spark Streaming 可以同时兼容批量和实时数据的处理逻辑，以便于历史数据的融合。Spark Streaming 的编程方式和 Spark 很类似，它通过 DStream 这种 RDD 序列来表示数据流并进行运算。

Spark Streaming 和 Storm 的相似点在于，它们的处理模式都像流水线作业一样，故称为流式计算。不过也存在不同之处。Spark Streaming 是处理某个时间段窗口内的事件，而 Storm 处理的是每次传入的单个事件，理论上 Spark Streaming 的延时性要比 Storm 略高。不过 RDD 的机制赋予了 Spark Streaming 更高的灵活性和容错性。在 Storm 中，每个单独的记录通过系统时必须被跟踪，这样 Storm 才能够保证每个记录至少被处理一次。但是在从错误中恢复过来的时候 Storm 允许出现重复记录，这就意味着某些状态可能被错误地更新多次。而 Spark Streaming 只需要在批量级别进行跟踪处理，即便一个节点发生故障，也可以有效地保证每个时间窗内的小量数据被完整地处理一次。

## 4.4 本章心得

“小明哥，这次学习内容强度也不小啊，继续深入了解了 Hadoop 的 MapReduce 计算框架，再加上 Hive 等脚本语言，算是对这个生态系统有了比较全面的认识。而 Spark 感觉是为了替代 MapReduce 而提出的优化方案，并形成了另一套生态系统，包括 Spark Streaming 和 Spark SQL 等。从处理的及时性上来看，MapReduce 适合离线批处理，Spark Streaming

和 Storm 则更适合实时计算。ActiveMQ 和 Kafka 这样的消息机制也可以提升批处理系统的实时性。”

“是的，大数据的技术非常丰富，也形成了不同的生态体系。要注意，不同领域的技术也是可以相互融合的。例如，ActiveMQ 和 Kafka 等消息机制可以与流式处理相结合，在确保系统稳定性的前提下进一步提升实时响应速度。而 Hive 同样可以架构在 Spark 之上。”

“好的，收到。听上去，有了获取、存储和处理，大数据的基本技术就介绍得差不多了？”

“不然，本章只讨论了应用中数据的基本处理过程和工具。在实际运用中，这些工具不仅仅是简单地进行数据统计、字段查询之类的操作，还可以帮助我们完成更为复杂的业务需求。这些需要更为精细的理论和模型支撑，包括搜索、推荐、广告、数据挖掘和机器学习等。在第 5 章和第 6 章中，将会进行详细的介绍。”

“大宝，到这里也讲了不少新知识，有空多多实践。”

“辛苦大哥了，今天请你喝咖啡吧。”

“来罐可乐就行了，听说最近他们家在搞大抽奖活动，最高奖送顶配的水果笔记本，看看我手气是否够好哦！”

## 4.5 参考资料

### □《Hadoop 实战》

作者：Chuck Lam

译者：韩冀中

不错的 Hadoop 入门图书，介绍了基本概念、系统架构、安装和使用。包括本章介绍的 MapReduce 计算框架。对 Pig 和 Hive 也进行了简单的介绍。

### □《Hadoop 实战（第 2 版）》

作者：陆嘉恒

本书除了对 Hadoop 有了详细的介绍，还涵盖了不少 Hadoop 生态中的其他内容，包括 HBase、Hive、Pig、Mahout，以及分布式协同系统 Zookeeper，知识点覆盖比较全面。

### □《Spark 大数据处理：技术、应用与性能优化》

作者：高彦杰

作者结合自己的实践经验和对 Spark 源代码的研究，从技术层面讲解了 Spark 的体系结构、安装与部署、开发环境搭建、计算模型等内容。然后从应用的角度讲解了一些简单的、有代表性的案例及性能优化。

### □《大数据 Spark 企业级实战》

作者：王家林

对 Spark 进行了比较全面和详细的介绍，包括对源码的一些分析，比较适合高级用户。

## □《Hive 编程指南》

作者：Edward Capriolo, Dean Wampler, Jason Rutherglen

译者：曹坤

介绍了 Hive 的背景、工作模型和 HiveQL 语法等。对于各种类 SQL 的语法进行了较为全面的解释，是一本很好的日常参考手册。

## □《Pig 编程指南》

作者：Alan Gates

译者：曹坤

为入门者讲述了 Apache Pig 的基础知识，同时也向有一定使用经验的高级用户介绍了更加综合全面的 Pig 重要特性，如 Pig Latin 脚本语言、控制台 shell 交互命令，等等。

## □《开源大数据分析引擎 Impala 实战》

作者：贾传青

介绍了开源大数据分析引擎 Impala 的背景、安装与配置、架构、性能优化等，以及一些相关的应用设计原则和应用案例。

## □《ActiveMQ in Action》

作者：Bruce Snyder, Rob Davies, Dejan Bosanac

ActiveMQ 的入门书籍，包含原理和编程范例。同时也介绍了 JMS 的理念和设计。

## □《Apache Kafka》

作者：Nishant Garg

本书介绍了如何创建 Kafka 集群，并通过实例展示如何开发定制化的消息生产者和消费者。

## □《Getting Started with Storm》

作者：Jonathan Leibusky, Gabriel Eisbrueb, Dario Simonassi

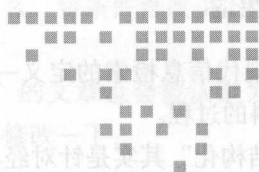
从零开始学习 Storm 的理念、架构和原理，适合初学者。

## □《Storm 分布式实时计算模式》

作者：P.Taylor Goetz, Brian O'Neil

译者：董昭

融合了作者的 Storm 实战经验，通过示例讲解使用 Storm 进行分布式实时计算的核心概念及应用，是一本不错的实践指南。



## 第5章

## Chapter 3

## 信息检索

“小明哥，前面已经讲解了大数据的平台和处理的相关知识。不过，我对网上的搜索功能是如何实现的也感到很好奇，看到只要输入几个简单的关键词就能返回用户想要的结果，这些仅仅靠数据的统计好像无法完成呢！”大宝提出了新的困惑。

“大宝，你很聪明啊，你的猜测很对。前面介绍了处理数据常用的方法和工具。可是，有些数据处理不再局限于统计下单词的频率，而是有更复杂的需求。例如：要在 100 万篇稿件中，查找讨论美食界最新动态的文章。基于传统的数据库很难高效地实现这一功能。在接下来的两个章节里，我们将分别介绍其中所涉及的两个关键处理技术：第 5 章介绍信息检索，侧重于信息的相关性和查询的高效性；第 6 章介绍数据挖掘，侧重于发现数据内部更深层次的价值。”

信息检索是个古老的话题，图书馆的管理一直使用着类似的技术。你去图书馆借书的时候，一定不会逐本地查看。更明智的方法是登录馆内信息系统，依照图书的大小分类，甚至是 ISBN 码找到对应的书架，这样查找的范围就会大幅缩小。随着互联网时代的到来，信息量爆炸，这项技术就更显得必不可少。新挑战、新需求使得检索技术被不断地革新。从 20 世纪 90 年代开始，逐渐形成了现代信息检索体系。相应地，也产生了一大批基于此的著名的互联网企业例如 Google、Amazon、eBay、百度、淘宝等。

本章将先介绍一下信息检索的一些基本理念和技术，以及相应的系统框架，最后探讨三个重要的子领域，包括搜索引擎、推荐系统和在线广告，其中介绍的开源系统包括 Lucene、Solr、Elasticsearch 和 Mahout，也可以让大家自己动手，尝试 DIY 搭建最简单的系统。



## 5.1 基本概念

学术界对现代信息检索的定义一般是：从大规模非结构化数据的集合中找出满足用户信息需求的资料的过程。

这里“非结构化”其实是针对经典的数据库而言的。数据库里的记录都有严格的字段定义 (Schema)，是“结构化”数据的典型代表。例如每道菜都有名字，想吃鱼时，查询“水煮鱼”就非常高效。相反，“非结构化”没有这种严格的定义，计算机世界存储的大量文本就是一个典型的代表。一篇文章如果没有经过特殊处理，对于其描述的菜叫什么名字、需要准备哪些食材等信息，我们是一无所知的。自然，我们也就无法将其中的内容和已经定义好的数据库字段进行匹配，所以这也是数据库在处理非结构化数据时非常乏力的原因。这时候就需要采用信息检索的技术来帮助我们。

如前所述，非结构化数据的特性是没有严格的数据格式定义，这点决定了信息检索的核心要素主要是如下两个方面。

□ 相关性：用户查询和返回结果之间是否有足够的相关性。缺乏了严格的定义，自然语言的表达含义是相当丰富的，如何让计算机更好地“理解”人类的信息需求是关键。

□ 及时性：用户输入查询后多久能获得返回的结果。要解决相关性，就要设计合理的模型，那么模型的复杂度可能会提升，相应计算量往往也会越高。因此需要高效率的系统实现。

## 5.2 相关性

相关性是个永恒的话题。“这篇文章是否和美食相关？”当被问及这个问题时，我们要大致看一下文章的内容，才能做出正确的判断。可是，至今为止计算机还无法真正懂得人类的语言，它们该如何判定呢？好在科学家们设计了很多模型，来帮助计算机处理基于文本的相关性<sup>①</sup>。

### 5.2.1 布尔模型

我非常喜欢吃川菜中的水煮鱼，其香辣的味道令人实在难忘。那么，如果我想看一篇介绍川菜文化的文章，最简单的方法莫过于看看其中是否提到了关键词“水煮鱼”。如果有（返回值为真），那么我认为就是相关的，如果没有（返回值为假），那么我就认为不相关。这就是最基本的布尔模型 (Boolean Model)。本次查询如果转化为布尔表达式也很简单，就一个条件：

水煮鱼

当然，你会觉得，中华饮食源远流长，川菜经典怎么会只有水煮鱼呢？没错，我还非

① 本节所提及的相关性，都是基于文本的数据。对图像、音像的处理不在讨论范围内。

常喜爱粉蒸排骨，它是小时候逢年过节的必备佳肴。那么，将条件修改一下：

水煮鱼 OR 粉蒸排骨

哈哈，这下，不仅仅是水煮鱼，谈到“粉蒸排骨”的文章也会被认定为相关。光说不练假把式啊，我们看看在上海哪里可以吃到它们呢？再修改一下：

(水煮鱼 OR 粉蒸排骨) AND 上海

这里，“上海”是必要条件，因为住在上海，不可能仅仅为了好吃的“打飞的”吧？然后水煮鱼或粉蒸排骨，有一样就可以啦，我是一个知足的男人。可是，我老婆看见了不乐意了，“我两样都想吃！”好吧，最后的布尔表达式修改如下：

(水煮鱼 AND 粉蒸排骨) AND 上海

最后，我们可以看看哪些文章提到了上海的哪些餐馆在经营川菜，而且同时还提供水煮鱼和粉蒸排骨两道佳肴。对，就这么简单，只要理解了布尔表达式，就能理解信息检索中的布尔模型。近些年，除了基本的 AND、OR 和 NOT 操作，布尔模型还有所扩展。其中，最常见的是邻近操作符 (Proximity)，用于确保关键词出现在一定的范围内。其假设就是：不同的搜索关键词，如果它们出现的位置越近，那么命中结果的相关性就越高。这里，为了提升相关性，再次添加“餐厅”这个关键词。不过“餐厅”和“上海”在文章中不一定是连续出现的，我们可以设置邻近操作符如下：

(水煮鱼 AND 粉蒸排骨) AND (上海 proximity 餐厅)

如果邻近的范围参数设置为 10，那么这条信息就是符合要求的：

……这家上海知名的餐厅，特色菜主要有粉蒸排骨、干锅牛蛙、水煮鱼、手撕包菜……

因为关键词“上海”和“餐厅”出现的位置间距没有超过 10 个词。

但如下的这条就不符合要求了：

……结束了上海的行程，我们来到了首都北京。这里的美食汇聚各地之大成。其中有名的川菜包括粉蒸排骨、干锅牛蛙、水煮鱼、手撕包菜，等等。我们迫不及待地挑选了一家当地知名的四川餐厅，放下了沉重的行囊……

“上海”和“餐厅”出现的位置间距远远超过了 10 个单词。

总体上来看，布尔模型的优点是简单易懂，系统实现的成本也较低。不过，它的弱点就是对相关性的刻画不足。相关与否是个模糊的概念，有的文章和查询条件关系密切，非常符合用户的信息需求，而有些则不尽然。仅仅通过“真”和“假”两个值来表示，不仅过于绝对，也没有办法体现其中的区别。那么，有没有更好的解决方案呢？

## 5.2.2 基于排序的布尔模型

为了增强布尔模型，需要考虑如何为匹配上的文档来打分。相关性越高的文档获得的分数就越高。第一个最直观的想法就是：每个词在不同文档里的权重是不一样的，可以通过这个来计算得分。这里介绍一下使用最为普遍的 *tf-idf* 机制。

假设我们有一个文档集合 (Collection),  $c$  表示整个集合,  $d$  表示其中的一篇文章,  $t$  表示一个单词。那么这里的  $tf$  就表示词频 (Term Frequency), 即一个词  $t$  在文章  $d$  中 (或是文章的某个字段中) 出现的次数。一般的假设是, 某个词在文章中的  $tf$  越高, 表示该词  $t$  对于该文档  $d$  而言越重要。当然, 篇幅更长的文档可能拥有更高的  $tf$  值, 我们会在后面 Lucene 的实现中讨论如何计算可以使得  $tf$  更合理。

同时, 另外一个常用的频率是  $idf$ , 它表示逆文档频率 (Inverse Document Frequency)。首先,  $df$  表示文档频率 (Document Frequency), 即文档集合  $c$  中出现某个词  $t$  的文档数量。一般的假设是, 某个词  $t$  在文档集合  $c$  中, 出现在越多的文档中, 那么其重要性就越低, 反之则越高。刚开始可能感觉有点困惑, 但是仔细想想不难理解。好比“的, 你, 我, 他, 是”这种词经常会出现出现在文档中, 但是不代表什么具体的含义。再举个例子, 在讨论美食的文档集合中, “美食”可能会出现在上万篇文章中, 但它并不能使得某篇文档变得特殊。相反, 如果只有 3 篇文章讨论到水煮鱼, 那么这 3 篇文章和水煮鱼的相关性就远远高于其他文章。“水煮鱼”这个词在文档集合中就应该拥有更高的权重。对此, 通常用  $df$  的反比例指标  $idf$  来表示, 基本公式如下:

$$idf = \log \frac{N}{df} \quad (5-1)$$

其中  $N$  是整个文档集中文章的数量,  $\log$  是为了确保  $idf$  分值不要远远高于  $tf$  而埋没  $tf$  的贡献, 默认取 10 为底。所以单词  $t$  的  $df$  越低, 其  $idf$  越高,  $t$  的重要性也就越高。那么综合起来,  $tf-idf$  的基本公式表示如下:

$$tf-idf = tf \times idf$$

也就是说, 一个单词  $t$ , 如果它在文档  $d$  中的词频  $tf$  越高, 且在整个集合中的  $idf$  也很高, 那么  $t$  对于  $d$  而言就越重要。

为了便于理解, 假设有一个关于美食的文档集合 (Collection), 其中包含了 10000 篇文章。其中一篇文章内容如下:

昨日, 王蓉来成都宣传新专辑《多爱》, 现场说起了四川话, 更哼起四川清音, 赢得一片叫好声。去年以一首《我不是黄蓉》走红歌坛后, 王蓉来蓉宣传专辑时, 终于感受到梦寐以求的川菜魅力。大快朵颐时, 王蓉突发灵感想到把川菜写进歌里。由于对水煮鱼感情最深厚, 王蓉提笔将水煮鱼写进了新歌, 唱出一个俏皮可爱、正处热恋期的女孩对水煮鱼、对热辣爱情的憧憬。

分别观察下面的 3 个词。因为不可能列出所有的文章, 假设这里的  $df$  统计都是正确的:

四川:

$tf = 2$

$df = 1200$

$$idf = \log \frac{10000}{1200} = 0.92$$

$$tf \times idf = 2 \times 0.92 = 1.84$$

水煮鱼:

$$tf = 3$$

$$df = 80$$

$$idf = \log \frac{10000}{80} = 2.10$$

$$tf \times idf = 3 \times 2.10 = 6.30$$

专辑:

$$tf = 2$$

$$df = 4$$

$$idf = \log \frac{10000}{4} = 3.40$$

$$tf \times idf = 2 \times 3.40 = 6.80$$

因为在讨论美食的集合中,很少有谈及娱乐圈的事情,这篇文章算是奇葩了,所以“专辑”这个词的  $idf$  分值非常高,  $tf-idf$  权重也就很高了。所以“专辑”一词对于这篇文章来说权重高于“四川”和“水煮鱼”。

除了考虑词的权重,还可以考虑不同字段的权重。例如,一篇文章既有标题,也有正文。那么一般情况下,我们会认为标题里出现的关键词更为重要,其查询匹配也应该赋予更多的权重,因此可以将标题和正文区分成两个不同的字段来对待。最基本的得分  $S$  应采用如下的线性加和来计算:

$$S = w_1 \times f_1 + w_2 \times f_2 + \cdots + w_n \times f_n \quad (5-2)$$

其中  $w_1$  到  $w_n$  分别为第 1 个字段到第  $n$  个字段的权重。而  $f_1$  到  $f_n$  分别为第 1 个字段到第  $n$  个字段中和查询关键词匹配的次数。

我们看下面这两个例子。

### 文章 1

标题:《川味水煮鱼》

正文:川味水煮鱼是川菜中常见的菜品,选鲈鱼、罗非鱼或鲢鱼,再配料蔬菜,以调料:姜、蒜、葱、豆瓣(或剁椒)、花椒、干红辣椒、辣椒粉、盐、味精、胡椒粉、料酒、酱油、醋少许、食用油、生粉、料酒、盐少许。

### 文章 2

标题:《王蓉成都宣传新专辑,〈多爱〉》



正文：昨日，王蓉来成都宣传新专辑《多爱》，现场说起了四川话，更哼起四川清音，赢得一片叫好声。去年以一首《我不是黄蓉》走红歌坛后，王蓉来蓉宣传专辑时，终于感受到梦寐以求的川菜魅力。大快朵颐时，王蓉突发灵感想到把川菜写进歌里。由于对水煮鱼感情最深厚，王蓉提笔将水煮鱼写进了新歌，唱出一个俏皮可爱、正处热恋期的女孩对水煮鱼、对热辣爱情的憧憬。

可以看到，虽然文章 1 中出现“水煮鱼”仅仅 1 次，不及文章 2 中出现的 3 次之多，但是文章 1 匹配的字段是标题，我们通常认为这种文章和“水煮鱼”的直接相关性更高。因此可以设置标题字段的权重是 10，而正文字段的权重仅为 1。那么文章 1 的得分  $s_1$  为：

$$s_1 = 10 \times 1 = 10$$

文章 2 的得分  $s_2$  为：

$$s_2 = 1 \times 3 = 3$$

至此，我们讨论了两种基本的打分机制，使得布尔模型也能获得不同的相关性得分，最后达到排序的功能。这里还可以结合词的权重和字段的权重，将公式 5-2 改良为如下形式：

$$S = w_1 \times ft\text{-}idf_1 + w_2 \times tf\text{-}idf_2 + \dots + w_n \times tf\text{-}idf_n$$

其中  $w_1$  到  $w_n$  依旧是第 1 个字段到第  $n$  个字段的权重。而将  $f_1$  到  $f_n$  替换为  $tf\text{-}idf_1$  到  $tf\text{-}idf_n$ ，代表第 1 个字段到第  $n$  个字段中查询关键词的  $tf\text{-}idf$  值。

### 5.2.3 向量空间模型

排序布尔模型是基于加权求和的打分机制，下面将介绍一个比排序布尔模型稍微复杂一点的向量空间模型（Vector Space Model, VSM）。此模型的重点，就是将某个文档转换为一个向量。还是以上面的文章 2 为例。

统计其中的单词，假设去重后一共有 50 个不同（Unique）的单词，那么该文档的向量就是 50 个维度，其中每个维度是其对应单词的  $tf\text{-}idf$  值，看上去就像这样：

（四川 = 1.84，水煮鱼 = 6.30，专辑 = 6.80，……）

当然，在系统实现的时候，不会直接用单词来代表维度，而是用单词的 ID，这点会在 5.3 节中的倒排索引中提及。如此一来，一个文档集合就会转换为一组向量，每个向量代表一篇文档。这种表示忽略了单词在文章中出现的顺序，可以大大简化很多模型中的计算复杂度，同时保证了相当的准确性，我们通常也将这种处理方式称为词包（Bag Of Word）。同理，用户输入的查询也能转换为一组向量，只是与文档向量相比较，查询向量会非常短。最后，相关性问题的转化就转化为计算查询向量和文档向量之间的相似度了。在实际处理中，最常用的相似性度量方式是余弦距离。因为它正好是一个介于 0 和 1 之间的数，如果向量一致就是 1，如果正交就是 0，符合相似度百分比的特性，具体的计算公式如下：

$$\text{Cosine: Sim}(d, q) = \frac{\sum_i (a_i \times b_i)}{\sqrt{\sum_i a_i^2 \times \sum_i b_i^2}} = \frac{(a_1 \times b_1 + a_2 \times b_2 + \dots + a_n \times b_n)}{\sqrt{(a_1 \times a_1 + \dots + a_n \times a_n) \times (b_1 \times b_1 + \dots + b_n \times b_n)}}$$

其图形化解释如图 5-1 所示。

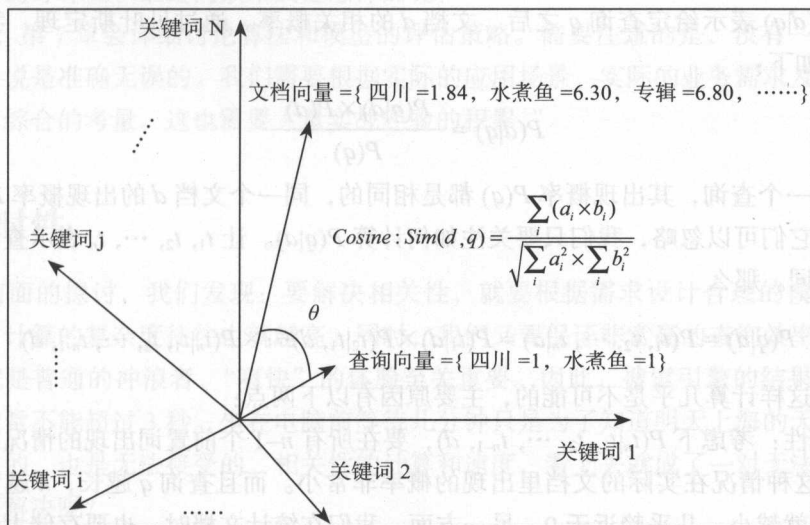


图 5-1 向量空间模型的夹角余弦 Cosine 计算图解

相对于标准的布尔数学模型，向量空间模型具有如下优点：

- 基于线性代数的简单模型，非常容易理解。
- 词组的权重可以不是二元的，例如采用 *tf-idf* 这种机制。
- 允许计算文档和索引之间的连续相似程度和基于此的排序，而不限于布尔模型的“真”、“假”两值。
- 允许关键词的部分匹配。

当然，基本的向量空间模型也有很多不足：例如对于很长的文档，相似度的得分不会理想；没有考虑到单词所代表的语义，还是限于精准匹配；没有考虑词在文档中出现的顺序等。

#### 5.2.4 语言模型

近几年来，另一个流行的检索模型是语言模型（Language Model）。语言模型本身并不是新兴的技术，它之前就已经在机器翻译、语音识别和中文分词中得到了成功的应用。从基本思路来说，布尔模型和向量空间检索模型都是从查询的角度出发的，观察查询和文档之间的匹配程度，并以此来决定如何找出相关的文档。然而，应用于信息检索里的语言模型是一种逆向思维：它为每个文档建立了不同的语言模型，用于判断由文档生成查询的概率是多

少，并将这个概率作为最终排序的依据。

为了更好地理解语言模型的这种逆向思维，先来看一下著名的贝叶斯定理。贝叶斯定理还在数据挖掘中的 Naive Bayes 分类器里起着关键作用，第 6 章也会提及。

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

假定  $P(d|q)$  表示给定查询  $q$  之后、文档  $d$  的相关概率。通过贝叶斯定理，我们可以将  $P(d|q)$  重写如下：

$$P(d|q) = \frac{P(q|d) \times P(d)}{P(q)}$$

对于同一个查询，其出现概率  $P(q)$  都是相同的，同一个文档  $d$  的出现概率  $P(d)$  也是固定的。因此它们可以忽略，我们只要关注如何计算  $P(q|d)$ 。让  $t_1, t_2, \dots, t_n$  表示查询  $q$  里包含的  $n$  个关键词。那么

$$P(q|d) = P(t_1, t_2, \dots, t_n|d) = P(t_1|d) \times P(t_2|t_1, d) \cdots \times P(t_n|t_1, t_2, \dots, t_{n-1}, d)$$

现实中这样计算几乎是不可能的，主要原因有以下两点：

□ 经济性：考虑下  $P(t_n|t_1, t_2, \dots, t_{n-1}, d)$ ，要在所有  $n-1$  个前置词出现的情况下，再出现  $t_n$ ，这种情况在实际的文档里出现的概率非常小。而且查询  $q$  越长， $n$  越大，这种可能性就越小，几乎趋近于 0。另一方面，我们在统计文档时，也要存储大量这种形式的数据，但是和查询又往往匹配不上，不会产生最终的效果，这对计算和存储都是一种极大的浪费。这也是我们通常所说的矩阵稀疏问题。

□ 查询随意性：用户在输入查询条件时，关键词之间的先后顺序通常都是随意的，因此这个顺序信息没有太高的价值。

所以在实践运用中最常见的还是一元和二元语言模型。

一元语言模型 (Unigram Language Model) 如下：

$$P(t_n|t_1, t_2, \dots, t_{n-1}|d) = P(t_1|d) \times P(t_2|d) \cdots \times P(t_n|d)$$

二元语言模型 (Bigram Language Model) 如下：

$$P(t_n|t_1, t_2, \dots, t_{n-1}|d) = P(t_1|d) \times P(t_2|t_1, d) \cdots \times P(t_n|t_{n-1}, d)$$

拿一元模型来说，我们可以看到最后相关性的问题就转化为了计算  $P(t_1|d)$  到  $P(t_n|d)$  的乘积。还是以前面的文章 2 为例。

假设分词后本文共 60 个词，那么“水煮鱼”和“四川”的概率分别为：

$$P(\text{水煮鱼}) = 3 / 60 = 0.050$$

$$P(\text{四川}) = 2 / 60 = 0.033$$

查询“四川水煮鱼”和该文档的相关性就为：

$$P(\text{四川}) \times P(\text{水煮鱼}) = 0.00165$$

考虑到概率乘积通常都非常小，在实际运用中还会通过一些数学手法来进行转换，但是原理保持不变。

“谢谢小明哥，现在，我对相关性的基本概念及主流的一些模型有了大致了解。那么如何评估模型的好坏呢？最终的效果又是怎样的呢？”

“别急，第7章会详细讨论算法和模型的评估策略。需要注意的是，没有一个模型是万能的，或者说是准确无误的。我们需要根据实际的应用场景、实际的业务需求及实际的数据质量，进行综合的考量。这也需要大量实战经验的积累。”

### 5.3 及时性

经过前面的探讨，我们发现：要解决相关性，就要根据需求设计合理的模型，越是精细的模型，计算的复杂度往往也就越高。同时，我们又要保证非常高的查询效率。互联网时代，用户就是普通的冲浪者，“爽快”的体验至关重要。因此，搜索引擎的结果处理必须是秒级的，通常不能超过3秒。坐在电脑前等待几分钟只是为了知道明天上海的天气情况，这是无法想象的，也是无法接受的。相关性的计算和速度，看上去就成了一对无法调和的矛盾体，该如何解决呢？

这里必须要提到搜索引擎最经典的数据结构设计——倒排索引（或者是逆向索引）。先让我们假想一下，你是一个热爱读书的人，当你进入图书馆或书店的时候，你会怎样快速发现自己喜爱的图书？没错，就是看书架上的标签。如果看到一个架子上标着“烹饪·地方美食”，那么恭喜你，离介绍川菜的书就不远了。倒排索引就是做着贴标签的事情。看看下面的例子（见表5-1），这是没有经过倒排索引处理的原始数据，当然现实中的文章不会如此之短。

表 5-1 五篇文章样例

文章 ID	内容
1	最上瘾的绝味川菜
2	大厨必读系列：经典川菜
3	舌尖上的川菜
4	舌尖上的中国味 在家吃遍八大菜系
5	舌尖上的历史

对于每篇文章的内容，我们先进行中文分词，然后将分好的词作为该篇的标签。例如对文章 ID 为 1 的文章“最上瘾的绝味川菜”进行分词，分为如下 5 个词<sup>①</sup>：

最，上瘾，的，绝味，川菜

<sup>①</sup> 你可能有不同的分法，这是因为中文分词本身也是门学问，5.5.4 节会略有介绍。



那么文章 1 就会有 5 个关键词标签，见表 5-2。

表 5-2 文章 1 分词之后的结果

关键词 ID	关键词	文章 ID
1	最	1
2	上瘾	1
3	的	1
4	绝味	1
5	川菜	1

再分析文章 ID 为 2 的文章“大厨必读系列：经典川菜”，它可以分为如下 5 个词：  
大厨，必读，系列，经典，川菜  
如果与第 1 次的标签结果合并起来，我们可以得到表 5-3。

表 5-3 文章 1 和文章 2 分词之后的结果

关键词 ID	关键词	文章 ID
1	最	1
2	上瘾	1
3	的	1
4	绝味	1
5	川菜	1, 2
6	大厨	2
7	必读	2
8	系列	2
9	经典	2

请注意关键词 ID 为 5 的关键词“川菜”，它在文章 1 和文章 2 中都出现过，所以我们会将其对应的文章 ID 写上“1, 2”。

如此逐个分析完所有五篇文章后，我们会得到表 5-4，这就是倒排索引的原型。

表 5-4 五篇文章分词之后所建立的倒排索引

关键词 ID	关键词	文章 ID
1	最	1
2	上瘾	1
3	的	1, 3, 4, 5
4	绝味	1
5	川菜	1, 2, 3
6	大厨	2
7	必读	2
8	系列	2

(续)

关键词 ID	关键词	文章 ID
9	经典	2
10	舌尖	3, 4, 5
11	上	3, 4, 5
12	中国	4
13	味	4
14	在家	4
15	吃遍	4
16	八大	4
17	菜系	4
18	历史	5

这里一共出现了 18 个不重复的单词，我们将这个集合称为文档集合的词典或词汇 (Vocabulary)。从这个结构可以看出，建立倒排索引的时候，是将文档 - 关键词的关系转变为关键词 - 文档集合的关系，同时逐步建立词典。有了这个，你会发现通过关键词查询起来就像在图书馆里根据书架上的标签找书一样方便快捷，效率得到大大提升。对于布尔模型而言，如上这种最简单的倒排索引完全可以满足需求。例如，我们查找：

川菜 AND 舌尖

通过查找“川菜”系统会返回文章 1, 2, 3；通过查找“舌尖”系统会返回文章 3, 4, 5。通过取交集，我们就能找到文章 3 并进行返回。取交集的归并操作在计算机领域已经非常成熟，速度也快得惊人。

考虑到布尔模型的邻近 (Proximity) 操作，我们还可以在数据结构中加上词的位置信息。假设词在文章的位置信息被标识为表 5-5 所示的形式。

表 5-5 分词的位置信息

最	上瘾	的	绝味	川菜
1	2	3	4	5

那么文章 1 的 5 个关键词标签可以如表 5-6 这样加上位置信息。

表 5-6 文章 1 分词之后加上位置信息

关键词 ID	关键词	文章 ID: 位置
1	最	1:1
2	上瘾	1:2
3	的	1:3
4	绝味	1:4
5	川菜	1:5

而最终的倒排索引看上去应该如表 5-7 所示。

表 5-7 五篇文章分词之后所建立的倒排索引（包含位置信息）

关键词 ID	关键词	文章 ID: 位置
1	最	1:1
2	上瘾	1:2
3	的	1:3, 3:3, 4:3, 5:3
4	绝味	1:4
5	川菜	1:5, 2:5, 3:4
6	大厨	2:1
7	必读	2:2
8	系列	2:3
9	经典	2:4
10	舌尖	3:1, 4:1, 5:1
11	上	3:2, 4:2, 5:2
12	中国	4:4
13	味	4:5
14	在家	4:6
15	吃遍	4:7
16	八大	4:8
17	菜系	4:9
18	历史	5:4

即便如此，你很快就能发现如果要通过其他模型进行计算，这些信息量仍然不够。例如向量空间模型，就还需要 *tf-idf* 的信息。这时，可以如表 5-8 一样来定义。

表 5-8 五篇文章分词之后所建立的倒排索引（包含 *tf-idf* 信息）

关键词 ID	关键词	关键词 <i>idf</i>	文章 ID: <i>tf</i> (由于文章太短, <i>tf</i> 全是 1)
1	最	0.699	1:1
2	上瘾	0.699	1:1
3	的	0.097	1:1, 3:1, 4:1, 5:1
4	绝味	0.699	1:1
5	川菜	0.222	1:1, 2:1, 3:1
6	大厨	0.699	2:1
7	必读	0.699	2:1
8	系列	0.699	2:1
9	经典	0.699	2:1
10	舌尖	0.222	3:1, 4:1, 5:1
11	上	0.222	3:1, 4:1, 5:1
12	中国	0.699	4:1

(续)

关键词 ID	关键词	关键词 <i>idf</i>	文章 ID: <i>tf</i> (由于文章太短, <i>tf</i> 全是 1)
13	味	0.699	4:1
14	在家	0.699	4:1
15	吃遍	0.699	4:1
16	八大	0.699	4:1
17	菜系	0.699	4:1
18	历史	0.699	5:1

其中 *idf* 的计算仍然使用的是公式 5-1, 文档集合总数  $N$  为 5,  $\log$  以 10 为底。同时, 这就意味着, 你可以设计一个更复杂的结构, 存储所有单词出现的位置、*tf*、*idf*、用于语言模型的概率, 甚至是其他附加信息, 在后面搜索引擎的实现部分也会做进一步讨论, 这里不再深入展开。这里只需要记住最重要的结论: 我们可以通过倒排索引保持超高的效率。谈到处理效率的问题, 第 7 章也会阐述如何对系统的性能进行评估。

## 5.4 与数据库查询的对比

之前阐述了信息检索的两大要素: 相关性的模型, 以及如何使用倒排索引的结构来提升相关性计算的效率。这里将通过对比信息检索和数据库查询的异同点来加深理解。先来看看相同点吧, 主要是:

- 擅长存储海量的数据。现代计算机的硬件处理能力日新月异, 即使是一台普通的笔记本电脑, 都能通过 SQL 数据库或搜索引擎存放上万甚至上亿条的数据记录。
- 支持关键词的查询。无论是数据库还是搜索引擎, 都允许用户输入需要查询的内容, 并且返回相应的结果, 完成较为实时的查询。

当然, 两者也有很多不同的地方, 主要是:

- 存放数据的格式不同。目前最为流行的关系型数据库, 存储的通常是格式化的关系型数据, 例如年龄、性别、身高等。而对搜索而言, 需要处理的非结构化数据往往更加多元, 除了文本, 还有图像、视频、音频, 甚至是用户等, 都可以纳入这个大的范畴。
- 查询的语言不同。使用数据库需要精通一种叫作 SQL 的专业语言, 需要一定的技术背景才能理解。所以数据库的直接用户大多数时候都是专业的程序员。而终端用户都是通过友好的界面来和数据库打交道的, 程序员的代码替他们完成了复杂的中间翻译工作。而搜索引擎通常采用的是布尔表达式 (Boolean Expression), 这种表达式对于非技术人员而言也是比较容易理解的, 因此程序员无须做太多的工作, 就能将终端用户的输入转变为引擎能看得懂的语言。
- 查询的实时性要求不同。如前所述, 数据库的查询通常是给专业人士, 或者是专业



系统使用的，很有可能是大规模的批量处理，因此对响应速度的要求并不高。对于数据分析师而言，等几分钟甚至更长的时间都是可以接受的。因此，数据库的任务也通常是离线运行的。而搜索则不同，使用者不可能等待过久。特别是互联网时代，用户就是普通的冲浪者，“爽快”的体验至关重要。因此，检索引擎的结果处理必须是秒级的，通常不能超过3秒。坐在电脑前等待几分钟只是为了知道明天上海的天气情况，这是无法想象的，也是无法接受的。

□ 索引类型不同。这个与上述的第3点是相关的。数据库完成的任务是通过精确的ID，查找关联的关系数据。例如输入学生的学号，查找他的身高体重和学分。这种顺序和存储的数据格式“学号、身高、体重、学分”是一致的，被我们称之为“正向索引”。而检索则不同，其主要的任务是通过关键词，反向查找哪些文章出现了这些词语。例如输入“中国互联网”，查找哪些文章是讨论这个话题的。数据库的正向索引是没办法及时返回结果的。因此必须使用一种之前介绍的“倒排索引”（或者“逆向索引”）的结构。

□ 数据更新的实时性不同。通过第3点和第4点，我们了解到相比数据库，检索引擎的查询实时性要求更高。正因为是正逆向的关系转换，所以必定会导致系统更新索引的开销要高于数据库。目前很多开源和商业的检索引擎在此方面做了不少优化，使得搜索索引的增量修改不再是瓶颈。但是，和传统的成熟数据库相比还是有一定的差距。

□ 评估的策略不同。数据库的查询都是精准匹配，因此评价的时候不用考虑准确性，而主要是看系统的性能。可是，信息检索需要解决的是用户主观而又模糊的信息需求。因此我们需要一套更复杂的理论和技术来评价“相关”与否，这点会在第7章具体介绍。

“既然如此，那么通常我们是用数据库比较好，还是搜索引擎比较好呢？”

“好问题！”小明不假思索地说道：“综合来说，两者并不存在孰优孰劣的问题，还是结合实际应用的场景，根据它们各自的特点来选择。正如刚刚的比较，数据库更适合结构化、关系型数据的精确查询，支持频繁地修改和删除之类的写操作，但不一定支持非常实时的查询。而搜索必须经过倒排索引的建立，对实时性强、模糊性强的查询非常有利，但是频繁地修改和删除的效率却不一定高，更适合以读取为主的应用。”

## 5.5 搜索引擎

互联网时代的到来，促使现代搜索引擎飞速的发展。目前，搜索引擎是信息检索最成功、最广泛的应用之一。它几乎有信息检索所有的要素：预处理文本信息、构建倒排索引、匹配查询关键词、计算相关性。但是，鉴于互联网应用的特殊性，它也有自身的独到之处。本节将重点介绍下搜索引擎的特殊性，对应的系统架构，以及成功的开源搜索引擎案例。

在传统的信息检索中，特别是文档搜索，相关性是至关重要的。可是，在互联网时代的各种应用中，除了相关性，我们还需要考虑其他的因素。例如，对于 Web 的搜索引擎，我们要考虑网页的权威性；对于电子商务而言，商品的热销度和价格等则是顾客非常关心的；对于在线广告而言，广告栏位的可信度也会影响用户的点击率。因此，在检索结果的相关性等同或者基本等同的前提下，我们还要考虑在应用场景下还会影响最终转化率的因素。接下来我们来看两个典型场景的例子。

### 5.5.1 Web 搜索中的链接分析

近 20 年里，互联网的搜索引擎飞速发展，日趋成熟。和传统的信息系统相比，Web 拥有海量的信息，用户会发现，要找到相关的内容并不一定很难，但是信息来源的质量却参差不齐。有的排版整齐，内容丰富、可信度靠，而其他则反之。人们开始考虑相关性之外的东西。首先，也是最直观的，就是 Web 网络里的链接关系。如果将每个网页当作图论里的一个节点，而网页间的超链接表示图论里的边，那么我们可以得到类似图 5-2 的网络图（Web Graph）。

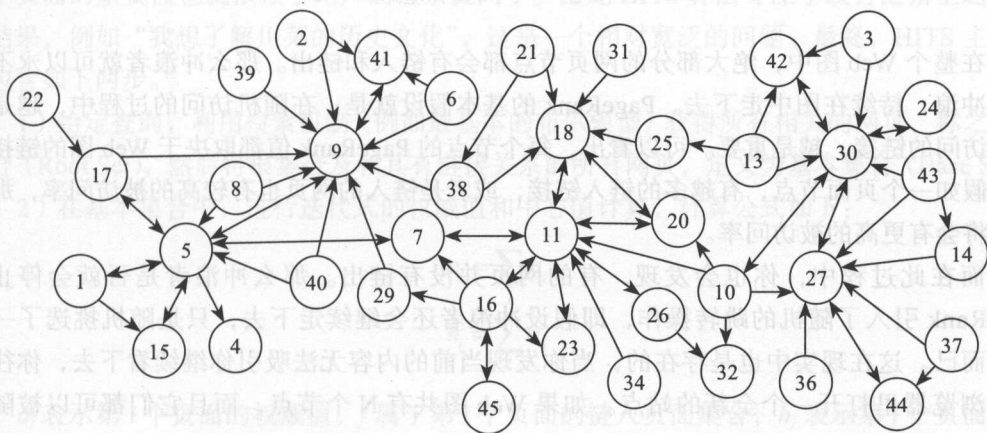


图 5-2 Web 形成的网络图样例

人们试图从这些图的模型中，得出有价值的线索，并用以确定信息的可靠性和质量。其中最著名的两个算法是 PageRank 和 HITS。

Google 是现代 Web 搜索引擎的重要代表。其巨大的成功离不开名为 PageRank 的核心算法。Google 的创始人拉里佩奇和谢尔盖布林于 1998 年发明了该项技术，并成为 Google 区别于当时搜索引擎的重要标志之一。其模型的假设是“随机冲浪者”，冲浪者从某张网页出发，根据 Web 图中的链接关系随机访问：在每一步中，他 / 她都会从当前网页的链出网页中随机选取一张作为下一步访问的目标。在图 5-3 中，一共有 5 个网页节点。网页 A 链接到 C 和 D，因此他 / 她会以 0.5 的概率（足够随机）分别访问这两张网页。假设网页 A 的

PageRank 值是 0.08，因此分别有 0.04 的 PageRank 值传递到 C 和 D。而网页 B 链接到 C、D 和 E，因此用户会以三分之一的概率分别访问这 3 张网页。假设网页 B 的 PageRank 值是 0.09，因此分别有 0.03 的 PageRank 值传递到 C、D 和 E。最终，C、D 和 E 的 PageRank 值分别为 0.07、0.07 和 0.03。

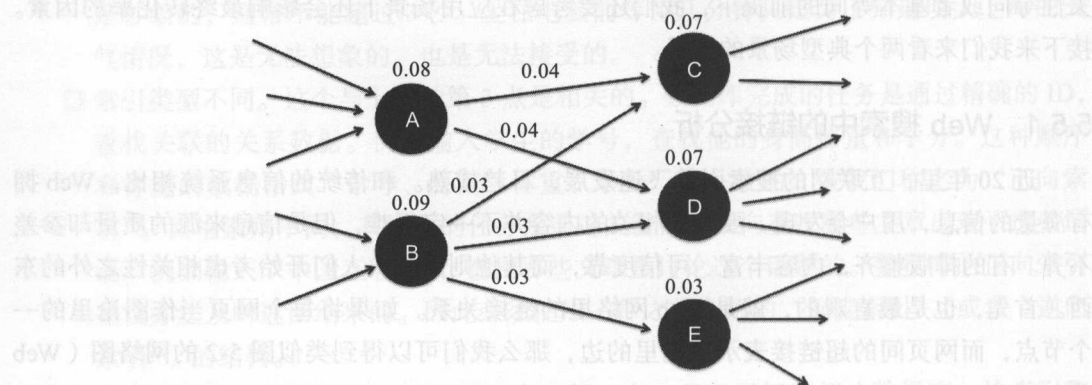


图 5-3 PageRank 随机冲浪模型示意

在整个 Web 图中，绝大部分的网页节点都会有链入和链出。那么冲浪者就可以永不停歇地冲浪，持续在图中走下去。PageRank 的基本假设就是：在随机访问的过程中，越是频繁访问的链接，越是重要。可以看出，每个节点的 PageRank 值都取决于 Web 图的链接结构。假如一个页面节点，有越多的链入链接，或者是链入的网页也有较高的被访问率，那么它也将会有更高的被访问率。

而在此过程中，你也会发现，有的网页并没有链出。那么冲浪者是否就会停止？PageRank 引入了随机的跳转操作，即假设冲浪者还会继续走下去，只是随机挑选了一张网页而已。这在现实中也是存在的，当你发现当前的内容无法吸引你继续看下去，你往往会在浏览器里打开一个全新的站点。如果 Web 图共有  $N$  个节点，而且它们都可以被随机跳转到，那么这个随机跳转的概率就是  $1/N$ 。综合一下，PageRank 最主要就是如下的两个步骤：

1) 当节点有出链接时，冲浪者将有  $w$  的概率 ( $0 < w < 1$ ) 随机跳转， $1-w$  的概率选择 1 个出链接进行下一步游走。其中会根据需求设置  $w$ ， $w$  越大就是假设用户越没有耐心，跳出当前页的可能性就越大。

2) 当节点没有出链接时，只能进行随机跳转。

可以用如下公式来表达这两个步骤：

$$PageRank(p_i) = \frac{1-w}{N} + w \times \sum_{p_j} \frac{PageRank(p_j)}{L(p_j)}$$

$p_i$  是当前计算 PageRank 值的页面， $p_j$  属于  $p_i$  链入页面的集合， $L(p_j)$  是  $p_j$  链出页面的



数量, 而  $N$  是整个页面集中所有页面的数量,  $w$  表示按照第 (1) 步游走的概率,  $(1-w)/N$  表示在整个集合中按照平均概率进行第 (2) 步的随机跳转。公式的内容很容易通过矩阵的迭代运算来实现。而且理论也证明, 以一定的初始值开始运算, 经过多次迭代后最终都可以达到一个收敛的状态。

除了 PageRank, 还有另一个有名的算法, 那就是 HITS, 它是由康奈尔大学 Jon Kleinberg 博士于 1998 年提出的。HITS 算法和 PageRank 最大的区别在于: 它将每张网页节点的值分为两个, 包括权威值 (Authority) 和中心值 (Hub)。Web 上有很多人工编辑的导航性网页, 例如各种网址导航, 其本身不会详细介绍某个主题, 但是会指向富含权威内容的网站, 这种节点称之为中心。相对而言, 如果一张网页本身不具备导航功能, 而是详细介绍某个主题的, 那么称之为权威。一张网页可以同时拥有中心和权威的身份, 只是得分有高低。HITS 假设一个优质的中心网页会指向更多优质的权威网页, 而一个优质的权威网页也会被更多中心网页所指向。因此, 我们可以采用类似于 PageRank 的方式给出中心值和权威值的循环定义, 并且通过迭代计算来求解, 最终达到收敛的状态。

和 PageRank 的另一个区别就是, Kleinberg 认为搜索是开始于用户的检索提问的, 那么每个页面的重要性也就依赖于用户的检索提问了。因此 HITS 算法专注于改善泛指主题检索的结果, 例如“我想了解川菜的历史文化”, 这是一个相对宽泛的问题。最终, HITS 主要步骤包含如下两步。

1) 给定查询, 利用检索机制 (例如最基本的布尔模型) 获得所有相关的网页, 称之为根集合 (Root Set), 然后将根集合及其有链接关系的所有网页, 定义为基本集合 (Base Set)。

2) 在基本集合中, 进行迭代式的权威值和中心值计算, 计算公式如下:

$$a_i = \sum_j h_j$$

$$h_i = \sum_j a_j$$

$a_i$  表示第  $i$  个页面的权威值,  $j$  属于第  $i$  个页面的链入页面集合,  $h_j$  表示第  $j$  个页面的中心值。类似地,  $h_i$  表示第  $i$  个页面的中心值,  $a_j$  表示第  $j$  个页面的权威值。

从上面的分析可以看出, PageRank 算法和 HITS 算法都是基于链接分析的分析算法。PageRank 算法的优点在于它对互联网上的网页给出了一个全局的重要性排序, 并且算法的计算过程是可以离线完成的, 这样有利于迅速响应用户的请求。不过, 其缺点在于主题无关性, 没有区分页面内的导航链接、广告链接和功能链接等。因此需要和其他相关性模型结合起来使用。HITS 算法的优点在于它只是对互联网中很小的一个相关子集进行分析, 所以它的相关性更有保障, 需要的迭代的次数更少, 收敛速度更快, 减小了时间复杂度。它的不足之处在于对于每个不同的查询算法都需要重新运行一次来获取结果。这使得它不可能用于实时系统, 因为对于上千万次的并发查询, 这样的开销实在太大了。



5.5.2 电子商务中的商品排序

近 10 年来，全球的电子商务呈现爆发式增长，成为互联网产业的一大支柱。随着商品的日益丰富，人们觉得要快速找到自己心仪的产品似乎越来越难，因此电商网站的搜索引擎也变得越来越重要。和其他网站有所不同，电商的搜索需要帮助用户挑选“最合适”的商品。除了基本的相关性，还涉及很多其他因素，例如价格、品牌、服务、口碑等。此外，由于存在不法商家用不正当手段牟利，还需要考虑反作弊等。图 5-4 列出了电商业界常用的金字塔模型。

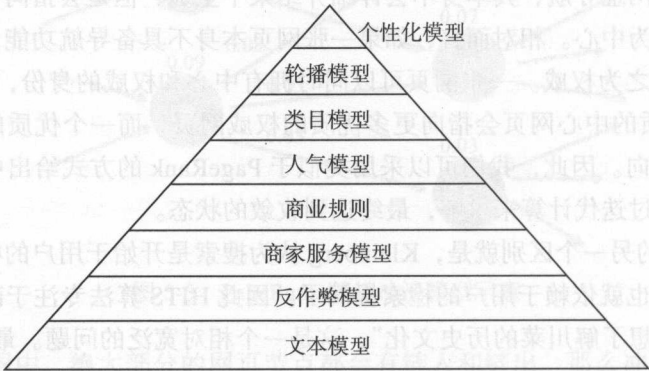


图 5-4 金字塔模型，从下往上权重逐步升高

1. 文本模型

统计商品信息中合理的关键词，这些和普通信息检索系统的实现类似。信息来源包括标题、副标题、品牌名、对应的类目路径名称、导购属性值，商家名称等。我们需要正确识别这些信息里的品牌词、产品词、型号词、规格词等。例如一个商品名称：

厨大哥 五香蒸肉粉调料 120G 清淡粉

文本模型要分析出来：

品牌词：厨大哥

产品词：蒸肉粉

口味词：五香、清淡

规格词：120G

2. 反作弊模型

作弊在常规信息检索系统里比较少见。但是到了互联网，特别是电子商务领域，由于涉及经济利益，营私舞弊就变得司空见惯。这也是该模型如此基础和重要的原因。我们需要识别不同的作弊手法，并对违规进行不同程度的处罚，最常见的手法有：

- ❑ 关键词作弊：标题、分类或属性中放入和商品本身无关的品牌和属性的词语。例如：比九阳更好的某某某牌豆浆机。那么当用户搜索“九阳豆浆机”时，该杂牌也会被

匹配上。

- 类目的故意错放：将商品放入热门但是没有关联的品类。例如：将一款面包机放入进口牛奶的类目。很多顾客都会买进口牛奶，浏览该类目的时候就会发现该面包机。
- 自卖自买的刷单：搜索排序的时候，销量是很关键的因素。自己买进自家的商品，是快速拉升排名的方法之一。
- 超低价格引流；和自卖自买类似，区别在于用一个超低的价格，吸引真实的顾客购买。当销量累积到相当的规模后，再恢复成正常甚至更昂贵的价格。如果搜索排名没有考虑销售额，这种方式也能拉升排名。
- 狸猫换太子：先用质量和价格诱人的商品打造爆款。在该商品排名靠前后，将其偷偷换成另一个商品的文描和价格，实现偷梁换柱。

电商的搜索系统需要一套控管系统，自动侦测可能的作弊行为，或者是接受运营人员的决策，对违反规定的商家进行处罚，例如搜索排名降权、沉底甚至是完全屏蔽处理。

### 3. 商家服务模型

除了商品本身，顾客还非常关心商家提供的售前、售中和售后的服务，这也超越了相关性的范畴。因此，电商搜索还需要衡量商家对消费者提供的综合服务质量。

商家质量主要指标有：

- 商家 DSR (Detailed Seller Rating, 最早起源于美国的 eBay 网站)。
- 店铺转化率，进入店铺的 UV (Unique Visitor) 或 PV (Page View) 访问转化为最终销售的比例。
- 店铺月动销率，店铺中有销售的品种在全部品种中的占比。

商家服务的主要指标有：

- 投诉率
- 退货率
- 24 小时发货率

### 4. 商业规则

根据市场战略，每个公司的商业规则也会有所不同。特定的情况下需要对某些具有特色服务的商家或商品给予一定的搜索排序、流量的倾斜等。例如让“超值”、“进口”、“新品”等特殊商品获得额外的排序加分。

### 5. 人气模型

这里统计的是用户对商品本身的认可，常见的主要指标包括：

- 近几天的销售金额(量)。一般是 7 天或 30 天。电商的销售除了搜索，还有很多其他渠道，例如促销活动、团购等，因此往往也会区分渠道，只计算搜索产生的销售。
- 转化率：现在主要是近  $n$  天的商品成交转化率，或者是商品浏览转化率，一般也是 7 天或 30 天。例如：商品成交转化率 = 搜索渠道近 7 天的订单 / 搜索页面近 7 天的曝

光次数。

□ 评分：电商网站通常也会提供用户评价和评分的功能。考虑到统计意义（这点在第7章中还会有介绍），可以同时考虑好评率和评价的数量。

□ UV / PV：基于商品的 UV（Unique Visitor）和 PV（Page View）的热度。

□ 收藏数：商品被用户收藏的数量。

## 6. 类目模型

根据商品发布时选择的分类和属性，以及用户搜索和点击的行为，确定某种分类商品的热度，在后面的实践章节中，可以看到其既可以用于相关性，也可以用于季节性和个性化等。例如，搜索“手机电池”推荐的是“手机电池”“移动电源”下的商品。假如发布商品时将其错放到“手机”类目下，将不能被正常推荐，排序会比较靠后。因此，我们需要结合反作弊模型来处理。

## 7. 轮播模型

该模型是出于流量分配的公平性考虑。根据商品的上架时间、曝光次数，对新品、低曝光高转化率的商品进行优先推荐。常用的方式是，首先离线统计出哪些商品满足新品、低曝光等条件，可以让其被轮播；一旦这些商品被轮播后其曝光率会增加，如果转化率不能保持较为良好的水准，那么之后其将不再会进入轮播队列。

## 8. 个性化模型

通过分析消费者的购物偏好，来给出定制化的排序。偏好包括但不限于：分类偏好、品牌偏好、属性偏好、合适价格区间等。个性化本身是一门很有意思的学问，不仅仅在搜索领域，还在推荐、广告、客户关系管理（CRM）等各个领域发挥着重要的作用。后面的实践篇也会谈到其具体的实现。同时，我们也要澄清一个误解：不少人将“个性化”和推荐系统等同起来。其实，个性化是一个宽泛的概念，在除了推荐以外的系统中，例如搜索，也是有其应用场景的。

### 5.5.3 多因素和基于学习的排序

通过前面两节的分析可以看出，在实际的搜索引擎应用中，相关性虽然很基础，但是还远远不够。例如，Web 搜索领域中的链接分析和电子商务领域的排序模型，就需要考虑很多其他的因素来确定最后的综合排名。那么问题也就随之而来了：

1) 如何结合不同的因素得到最终的排序得分？

2) 如何确定不同因素的权重？

第1个问题可以采用最简单的方式——线性加和。这点已经在5.2.2节基于排序的布尔模型中介绍过了，最终的总得分  $S$  可以按照如下公式来计算：

$$S = w_1 \times score_1 + w_2 \times score_2 + \cdots + w_n \times score_n$$



其中  $score_1$  到  $score_n$  对应于  $n$  个因素的得分。 $w_1$  到  $w_n$  是对应的  $n$  项权重。但是需要注意的是,这和基于排序的布尔模型有所不同:不同维度的因素取值范围可能会不一样。有的可能是从 0 到 1 之间的小数,有的可能是从 1 到 100 之间的整数。这时候需要使用数值的归一化 (Normalization), 将所有的取值统一到相同的取值空间。最简单的归一化就是使用线性的值变换, 比较复杂的还需要考虑取值的空间分布, 保证线性加和时的可比性。

针对第 2 个问题, 当因素不多的时候, 可以根据经验, 比较容易地定义不同因素的权重。可是, 随着业务需求和系统复杂程度的不断增加, 数百个因素是司空见惯的事情。全部通过人工来设置不太可能, 也未必合理。此时, 我们可以借助机器学习的算法, 来获取较优的权重取值。首先, 根据历史记录来挑选排序, 将靠前的物品作为训练的样本, 将其所有因素的数据记录下来作为输入值, 将用户是否选购记录下来作为目标值。然后采用回归技术<sup>①</sup>, 确定对于理想的物品, 其每个因素应该有怎样的权重取值。最终, 这些取值将确定在未来排序时, 不同的因素对决策的贡献程度。这种基于学习的方式, 在越来越多的搜索引擎中开始被使用, 在下一章介绍数据挖掘的相关知识时, 还会对回归技术进行更为详细的介绍。

“小明哥, 这里提到重要性是利用多种因子综合来实现的, 并且可以通过机器学习的方式确定不同因子的权重。那么, 5.2.1 节探讨的相关性是否也可以通过多种因子来实现呢?”

“这是个好问题, 你觉得呢? 如果可以, 你准备采用哪些因子呢?”

## 5.5.4 系统框架

在纵览了信息检索和搜索系统的主要特性后, 这里先对一般情况下实现搜索引擎的系统框架做个总结。倒排索引使得快速查询成为可能, 但是需要额外的预处理工作, 例如中文分词、建立并存储索引表等。考虑到所有的这些特性, 通常搜索系统的框架都被划分为两个重要步骤: 离线预处理和在线查询。

### 1. 离线预处理

通常包括数据获取、文本预处理、词典 (特征空间) 和倒排索引的构建、基本信息的统计等。

#### (1) 数据获取

对于常规的 Web 网络搜索, 发现并获取外部的网页信息是必不可少的步骤。通常, 这个步骤是通过第 1 章介绍的网页抓取爬虫 (Crawler/Spider) 来实现的。爬虫 Spider 顺着网页中的超链接, 从这个网站爬到另一个网站, 然后通过超链接分析连续访问并抓取更多的网页。被抓取的网页被称之为网页快照。由于互联网中超链接的应用很普遍, 理论上, 从一定范围的网页出发, 就能搜集到绝大多数的网页。这些在第 2 章已经有所讨论。需要注意的是, 并非所有的搜索应用都需要这一步, 例如电子商务的商品数据, 都是来自内部业务人员

<sup>①</sup> 例如用最常见的小二乘法来做线性回归。



的输入和运营。

## （2）文本预处理

常规的文本预处理是指针对中文等语系进行分词操作、针对英文的词干（Stemming）和归一化（Normalization）处理，以及所有语言都会碰到的停用词（Stopword）、同义词和扩展词处理。

□ 中文分词：中文比较复杂的地方在于分词。我们知道，在英文的行文中，单词之间是以空格作为自然分界符的，而中文只是字、句和段能通过明显的分界符来简单划界，唯独词没有一个形式上的分界符。中文分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。百度公司曾经播放过一则广告，很有创意，让我至今记忆犹新：两个秀才，在城门口比才艺，对同一副对联做出了完全不同的解释。广告的创意是体现百度对中文分词技术的深入研究，这也从侧面也反映出中文分词的难度。目前主流的分词模型分为两大类：

- 基于字符串匹配，即扫描字符串，如果发现字符串的子串和词相同，就算匹配。匹配规则通常是“正向最大匹配”、“逆向最大匹配”、“长词优先”。这些算法的优点是计算复杂度低，缺点是处理歧义词效果不佳。
- 基于统计和机器学习，这类分词基于人工标注的词性和统计特征，对中文进行建模。训练阶段，根据标注好的语料对模型参数进行估计。在分词阶段再通过模型计算各种分词出现的概率，将概率最大的分词结果作为最终结果。常见的序列标注模型有隐马尔科夫模型（HMM）和条件随机场（CRF）。

□ 词干和归一化：英文相对于中文而言，完全无须考虑分词。不过它有中文所不具备的单复数、各种时态，因此它需要考虑词干。词干还原的目标就是为了减少词的变化形式，将派生词转化为基本形式，例如：

am, are, is, was, were 全部转换为 be

car, cars, car's, cars' 全部转换为 car

最后，还要考虑大小写转化和多种拼写（例如 color 和 colour）这样的统一化，学术上称之为归一化。

□ 停用词：无论何种语言，都会存在对相关性判定意义不大的词，例如在 5.2 节介绍的 *idf* 很低的值。有的时候干脆可以指定一个叫停用词的字典，直接将这些词过滤，不将它们建入索引中。例如英文中的 a、an、the、that、is、good、bad 等。中文的“的、个、你、我、他、好、坏”等。如此一来，我们可以压缩索引文件的大小，在不损失甚至是提升相关性的前提下，提高查询的效率。当然，也要注意停用词的使用场景，例如后面的应用篇提到的用户观点分析，good 和 bad 这样的形容词反而成为关键。不仅不能过滤，反而要加大它们的权重。

□ 同义词和扩展词：不同的地域或不同的时代，会导致人们对于同样的物品叫法不相同。我当年在北京工作的时候，刚开始点菜时都会说来个“番茄蛋汤”，有些服务员

会稍微愣一下。没错，在中国北方“番茄”应该叫“西红柿”。对于检索系统而言，需要意识到这两个词是等价的。添加同义词就是一个很好的手段。我们可以维护如下一个同义词的词典：

番茄，西红柿，Tomato

洋山芋，土豆

泡面，方便面，速食面，快餐面

山芋，红薯

鼠标，滑鼠

.....

有了这样的词典，在离线处理的时候，系统看到文档中的“番茄”关键词，在索引里同时会增加“西红柿”这个词。如此一来，即使北方的用户查询“西红柿”也能匹配上该文档。

有的时候我们还需要扩展词。如果简单地将 Dove 分别和多芬、德芙简单地等价，那么多芬和德芙也变成了同义词，这样很明显是有问题的。那么我们可以采用偏序的扩展关系，当系统看到文档中的“多芬”时添加“Dove”，看到“德芙”时也添加“Dove”。但是看到“Dove”的时候不添加“多芬”或“德芙”。这样搜索“Dove”的时候就能同时查到多芬和德芙品牌的商品，而搜索多芬则不会误出现德芙巧克力。

这时，好奇的大宝头脑里又闪过一个想法：“同义词和扩展词，是否一定要在离线的时候预处理？可不可以在线查询的时候动态处理呢？”

小明反问道：“你觉得呢？如果可以实现，离线处理和在线处理各有什么优劣？”

### (3) 特征空间和倒排索引的构建

前面倒排索引部分介绍了将文集转换为关键词-文档的同时，可以构建该文集的词典 (Vocabulary)。而在向量空间模型 VSM 中，更是将词典中的每个单词作为向量的一个维度。其实，我们大可不必限制每个维度必须为单词。例如电子商务领域中，一个商品的用户购买数据，就不是代表某个单词，而是代表某个用户购买该商品的行为。其数值也不是 *tf-idf* 这种对词的权重评估，而是一定周期内用户购买的次数。因此可以将单词扩展为特征 (Feature)，相应的，字典扩展为特征空间 (Feature Space)。特征空间在信息检索和数据挖掘的领域中，都是非常重要的概念，这点在数据挖掘的章节还会提及。有了特征空间，我们就能利用散列表的结构来建造倒排索引。散列的键值是特征 ID，其后的链表存储了相应的数据。例如，对于粉蒸排骨这个商品，用户购买特征的 ID 为 1000，对应链表有 30 个用户节点，每个节点的 ID 是用户 ID，而数值是一年内该用户购买此粉蒸排骨的次数。如果我们再做进一步扩展，其实搜索检索的也不再限于文档，而是任何一个可以用特征来表示的数据对象，例如一名顾客，他/她可以用上网购物站点、次数、金额等来表示。

### (4) 应用相关的统计和排序

正如 5.5.1 至 5.5.3 节所介绍的，不同领域的搜索应用，需要考虑更多的因素。绝大部分的相关统计，都需要离线的收集、计算和存储。还有基于学习的排序，训练部分也是典型

的离线处理模块。

2. 在线查询

其实，对数据的离线处理完毕后，在线查询是相对简单和直观的。查询一般都会使用和离线模块一样的预处理，特征空间也是沿用离线处理的结果。当然，也可能会出现离线处理中未曾出现过的新特征，一般会忽略或给予非常小的权重。在此基础上，系统根据用户输入的查询条件，在索引库中快速检出数据对象，并给出相关度评价。例如：最简单的布尔模型只需要计算若干匹配条件的交集即可；向量空间的 VSM 模型只需要计算查询向量和待查向量的余弦夹角；语言模型只需要计算匹配条件的贝叶斯概率。

综上所述，可以得到如图 5-5 所示的概览图。

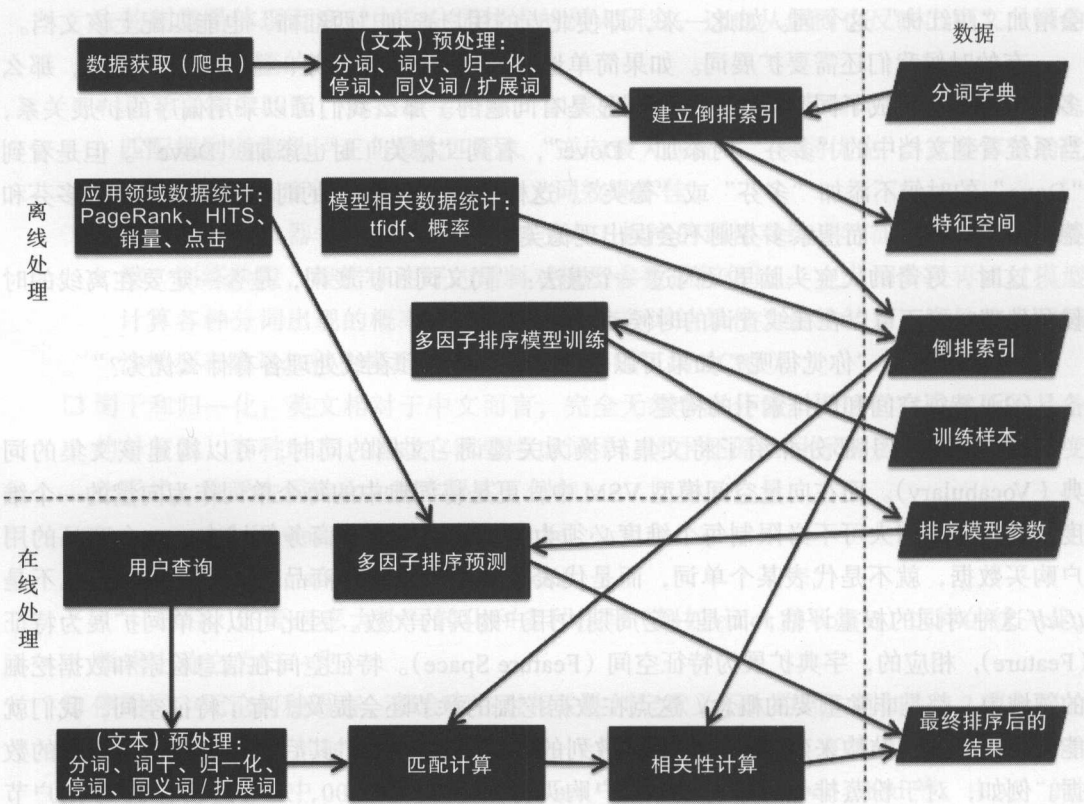


图 5-5 搜索引擎常见的系统架构

由于搜索引擎在大数据领域有着非常广泛的应用，不少开源 (Open Source) 的项目都对其提供了良好的实现。这些项目都有清晰的文档说明，而且更为重要的是，其良好的开放性，为我们进一步自定义扩展提供了广阔的舞台。目前非常流行的 3 个项目有 Lucene、Solr 和 Elasticsearch。Lucene (官方主页: <http://lucene.apache.org/>) 是 Java 家族中最为出名的一



个开源搜索引擎，在 Java 世界中已经是标准的全文检索程序了，它提供了完整的索引引擎和查询引擎。Solr（官方主页：<http://lucene.apache.org/solr/>）是一个用 Java 开发的独立的企业级搜索应用服务器，它对 Lucene 进行了良好的封装，提供了更丰富的功能和接口，对于企业级应用而言是一种更为成熟的解决方案。Elasticsearch（官方主页：<http://www.elasticsearch.org/>）和 Solr 类似，同样是一个采用 Java 语言、基于 Lucene 构造的开源的分布式搜索引擎。其设计目的是能够达到实时搜索、稳定可靠。下面来快速地介绍一下这三个开源系统。

### 5.5.5 Lucene 简介

Lucene（<http://lucene.apache.org>）是一个开放源代码的全文检索引擎工具包，是近几年非常受欢迎的免费 Java 信息检索程序库。它最初是由 Doug Cutting 编写的，并于 2000 年 3 月在 SourceForge 上开源并提供下载。2000 年 10 月发布了 1.0 版本，2001 年 7 月发布了最后一个 SourceForge 版本 1.01b。2001 年 9 月 Lucene 作为高质量的 Java 开源软件产品加入了 Apache 软件基金会的 Jakarta 家族，2005 年升级成为 Apache 顶级项目。Lucene 本身并不是一个完整的全文检索引擎，但是它提供了完整的索引引擎和查询引擎，部分文本分析引擎（英文等西方语言）。其目的是为软件开发人员提供一个简单易用的工具包，从而可以很方便地在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。

首先，让我们认识一下 Lucene 里面的两个重要的概念：字段（Field）和文档（Document）。

这里的“字段”和数据库里的概念非常类似，它包含了需要被搜索到的内容。例如文章的标题就是一个字段。Lucene 处理每个字段的方式也可以不同，有不同的选项可供选择，最基本的包括如何索引、是否存储、词条向量（TermVector）等。

其常见索引选项（这里的索引特指倒排索引）如下。

- ❑ Index.ANALYZED：将字段值分解成独立的单词流，并记入倒排索引，使得每个单词都能被搜索。这是最常规的选择，最典型的应用就是中文分词和英文抽取词干，适合如标题、正文等这样的普通文本。
- ❑ Index.NOT\_ANALYZED：不做任何分词、抽取词干和过滤停用词等操作，直接将字段值记入索引。适合电话号码、文件路径、网页 URL 等这样的“精确匹配”。这些信息如果采用 Index.ANALYZED，可能会导致命中错误的查询。
- ❑ Index.NO：字段值不记入索引，也无法搜索到。你可能会觉得奇怪，既然无法搜索，为什么还需要这个字段呢？这是因为，在特定场合下我们虽然不希望某个字段被查询到，但是还是希望其信息能被展示出来。例如：商品图片在服务器上的存放地址。这个地址普通的顾客当然无法知道，也是不需要知道的，所以无须作为搜索条件，自然也不用记入倒排索引。但是，当该商品被查询出来，并需要展示给顾客时，其图片信息还是必不可少的，因此有必要存储在正向索引里。下面将解释字段存储有哪些选择。

常见存储选项如下：



- ❑ Store.YES：将字段值进行存储。原始的值将全部存储在正向索引中，上述的商品图片就是一个很适合的应用场景。需要注意的是，相对上述的索引选型，这种操作会消耗较大的存储空间。在实践中我们并不鼓励将不必要的信息全部都存储在 Lucene 的索引中。这样不仅会浪费存储资源，也会使得在线查询变得缓慢。
- ❑ Store.NO：不存储字段值。如果一个字段既不倒排索引，也不正向存储，那么它就完全失去了意义。因此，如果选择了 Store.NO，那就必须要和 Index.ANALYZED 或 Index.NOT\_ANALYZED 共同使用。这样的好处在于既可以满足查询的需求，又可以节约硬件资源。

现在的搜索引擎，都可以返回和查询相关的原文片段，并将匹配的关键词用不同的背景色高亮显示，如图 5-6 所示。Lucene 同样可以实现这个功能，只是需要存储更多的信息。词条向量就是为了达到这个目的，而采用的一种中间数据结构。如果需要用到高亮功能，就需要打开 WITH\_POSITIONS\_OFFSETS。

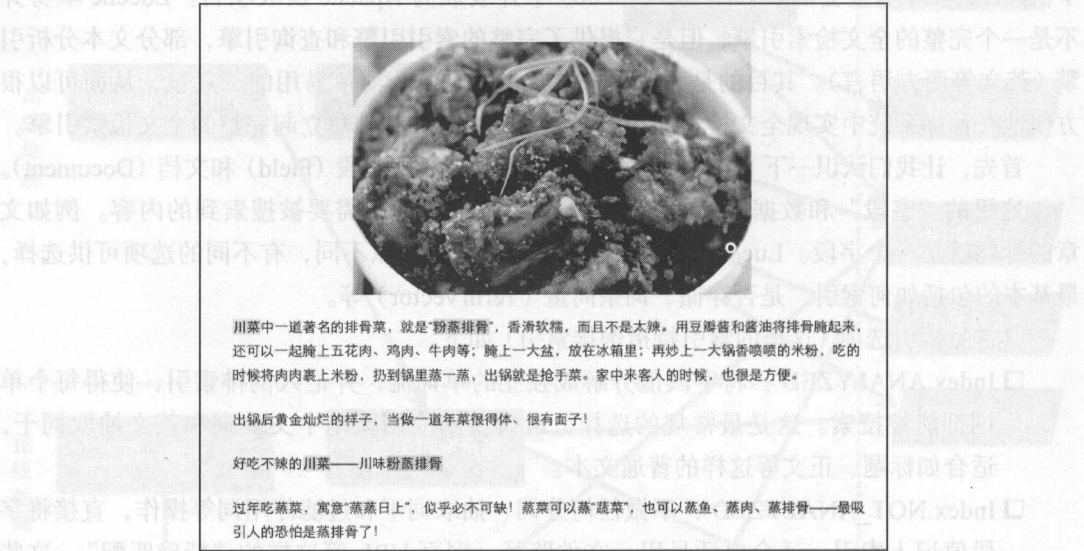


图 5-6 高亮显示搜索引擎中的关键词

表 5-9 是常用的选项组合及其应用场景。

表 5-9 选项组合和对应的场景

索引	存储	词条向量	应用场景
ANALYZED	YES	WITH_POSITIONS_OFFSETS	文档的标题和摘要
ANALYZED	NO	WITH_POSITIONS_OFFSETS	文档的正文
NO	YES	NO	文档配图的链接
NOT_ANALYZED	YES	NO	文档的作者和 ID

另外，在 Lucene 中，某个字段也可以包含多个值。比如，一篇文章的作者可能有多名，你完全没有必要为了 5 个作者而定义 5 个不同的字段。相反，你只需要往同一个字段中添加多个值，而且查询时这些值返回的顺序也将保持它们在索引添加时的顺序，一切就这么简单。

“小明哥，那么多值字段为什么不合并在一起成为一个值？感觉多值字段用处不明显啊。”

“大宝，你仔细想想，这里多个字段若合并在一起了，和多值字段相比有没有查询效果、存储及性能上的区别？”

介绍完字段，再来看下文档 (Document)。文档是 Lucene 索引和查询的最基本单位，它包含一个或多个字段。例如一篇文章可以包括 4 个字段：标题、正文、作者和日期。之所以使用“文档”这个名词，主要是因为 Lucene 处理的对象一开始都是文档集合。值得一提的是，随着搜索引擎的应用越来越广泛，被索引和搜索的主体也就不再局限于文档了。只要是能通过文本字符和数字表示的数据，都可以成为 Lucene 的处理对象。举个例子，我们可以将网站的用户作为一个文档，而将用户名、年龄、性别、职业、兴趣爱好、发表的评论作为 6 个字段。这样，Lucene 就能帮助我们轻松地实现对用户及其相关信息的搜索。在后面提及的 Solr 和 Elasticsearch 中，文档同样是宽泛的概念。

在了解完 Lucene 中的字段和文档的概念后，我们来看看 Lucene 是如何实现相关性和倒排索引这两个核心要素的。首先是相关性衡量，Lucene 默认的相关性得分计算大体上是通过对向量空间模型 VSM 来实现的，又融合了一些启发式的规则，具体公式如下：

$$\sum_{t \in q} (tf(t \text{ in } d) \times idf(t)^2 \times boost(t, \text{field in } d) \times lengthNorm(t, \text{field in } d)) \times coord(q, d) \times queryNorm(q)$$

表 5-10 中列出了公式各个部分所表示的含义。

表 5-10 公式各部分的含义

$tf(t \text{ in } d)$	单词 $t$ 的词频，即 $t$ 在文档 $d$ 中出现的频率
$idf(t)$	单词 $t$ 在文档集合中的逆文档频率，用来衡量 $t$ 在整个集合中的“唯一”性
$boost(t, \text{field in } d)$	字段和文档的加权。在离线的索引阶段，可以针对某个字段和文档进行加权
$lengthNorm(t, \text{field in } d)$	字段归一化的值。如果字段越短，该值就越大，获得的加权也越大
$coord(q, d)$	协调因子，命中的查询条件越多，该值就越大，获得的加权也越大
$queryNorm(q)$	查询归一化的值。区别不同查询条件的权重

其中  $tf$  和  $idf$  的机制，在 5.2.2 节介绍排序模型时有过介绍，大家可以回顾一下。Boost 实现了 5.2.2 节中所阐述的不同字段的加权功能。除了这些基本的要素，Lucene 还引入了下面几个关键值：

□ 字段归一化 ( $lengthNorm$ )。因为 Lucene 允许针对不同的字段进行查询，那么在标题里命中“水煮鱼”和在长篇大论里命中“水煮鱼”的效果肯定是不一样的。Lucene 里会设定字段长度越短，相关性越高。因此，如果有文档在标题里命中关键词，那么它的搜索排名肯定是高于其他在正文里命中关键词的文章。

□ 协调因子 (coord)。对于有多个查询关键词 (或条件) 的搜索, Lucene 会假设命中的关键词越多, 相关性越高。例如, 搜索“上海 水煮鱼 餐厅”, 文档“上海有哪些餐厅的水煮鱼味道很棒”会匹配上全部 3 个关键词, 而“川味水煮鱼的由来”只匹配上 1 个关键词。这样前一篇的排名应该更高。

□ 查询归一化 (queryNorm)。Lucene 假设包含多个条件的查询中, 每个条件的权重可以不一样。例如, 在搜索“上海 水煮鱼 餐厅”时, 我们设定“水煮鱼”是重要性最高的条件。那么“川味水煮鱼的由来”的排名可能会比“上海有哪些餐厅的水煮鱼味道很棒”更高。

从 4.0 版本开始, Lucene 就将排序相关的算法与向量空间模型解耦, 提供了其他模型的基本实现, 包括最佳匹配 Okapi BM 25、随机分歧 (Divergence from Randomness)、语言模型和基于信息量的模型等。当然, 最重要的一点是: 不要忘记 Lucene 是开源的哦。这就意味着我们可以根据需要自行修改这些计算逻辑, 甚至是完全实现一套新的模型。

除了相关性, 另一个核心要素就是倒排索引。下面来看看 Lucene 有哪些重要模块涉及倒排索引的构建和查询。

首先看离线部分。

□ 分析器 (Analyzer): 如果一个字段设置了 ANALYZED, 那么其值在被索引之前, 都会经过分析器的处理。它负责从文本中提取单词, 增加同义 / 扩展词, 过滤掉停用词等无用信息。还有一些基于特定语言的操作也会在此完成。例如英文中的词干抽取、归一化, 中文的分词等。Lucene 对拉丁语系支持较好, 自带不少分析器而且默认功能较为齐全, 但是对中文分词支持较弱。幸运的是, Lucene 有良好的开放性, 建议可以考虑集成 IKAnalyzer、ANSJ 这样的开源分词包。需要说明的是, 分析器不仅仅能在离线部分使用, 在线搜索时, 查询的分析也需要它。通常, 我们也需要保持离线索引和在线查询的分析器一致, 以免出现无法匹配的尴尬。

□ 索引器 (IndexWriter): 这是倒排索引过程中的核心组件, 负责创建新索引或打开已有的索引, 以及向索引中添加、删除或更新文档。在这个过程中, 也会根据字段的选项设定来决定每个字段值是否被索引、是否被存储等。

接着来看看在线部分。

□ 查询解析器 (QueryParser): 如 5.2 节所述的, 布尔表达式是构建查询语句的基础。可是, 随着应用需求的日益复杂, 写一个超长的表达式对于普通人而言实在是太痛苦了。Lucene 的查询解析器给使用者们带来了福音, 它允许用户使用形式更为自由的查询语言。看看这个例子吧——“上海 AND 餐厅 AND 酸菜鱼 AND (粉蒸排骨 OR 干锅牛蛙 OR 手撕包菜)”。对于查询解析器而言, “+ 上海 + 餐厅 + 酸菜鱼 + (粉蒸排骨 干锅牛蛙 手撕包菜)”, 这种简单的表述就能接受, 它会自动地替你转换为最终的表达式。当然, 查询解析器的功能非常强大, 还可以支持指定字段查询、动态增强得分、模糊匹配等。



❑ 搜索器 (IndexSearcher): 搜索器以只读的方式, 打开由索引器构建的索引, 并进行查询, 计算相关性得分。通常情况下只需要一个搜索器的实例就能满足所有的搜索请求。只在索引有所更新时, 才需要重新打开新的搜索器实例。

Lucene 之所以如此流行, 是因为除了上述搜索引擎的基本要素外, 它还有不少扩展的功能。这里列举几个主要的。

### (1) 多样化查询

❑ 范围查询: 除了文本, Lucene 还可以针对数字和日期设置索引, 并且针对它们提供的范围进行查询。例如查询阅读量超过 1 万的博客文章; 或者查询价格在 100 元到 1000 元之间, 并且 3 天内就要下架的商品, 诸如此类。

❑ 词组查询: 还记得布尔模型里的邻近操作吗 (Proximity)? 词组查询是一种实现方式, 而且我们可以设置邻近的单词间距。

❑ 前缀查询: 使用指定的开头字符串作为搜索条件, 然后查询匹配的文档。

❑ 通配查询: 使用包含通配符的字符串作为搜索条件。Lucene 常用的通配符: \* 代表 0 个或多个字母, ? 代表 0 个或 1 个字母。例如, 搜索“酸\*”, 酸菜、酸豆角、酸不溜秋都会匹配上。

❑ 模糊查询: 根据字符串的编辑距离 (Edit Distance), 支持单词部分匹配的查询。

### (2) 过滤查询

过滤查询是 Lucene 用于缩小搜索空间的机制。和普通查询有所不同, 过滤查询不会计算相关性得分。也正是如此, 其计算速度要快于普通查询。所以, 它适用于无须考虑相关性的场景。例如, 我们要查找性别为男性、年龄在 25 岁和 50 岁之间、家有子女的用户。这里只需要考虑“满足”或“不满足”三个过滤条件即可。

### (3) 高亮显示

Lucene 提供的高亮模块能够拆分和高亮显示查询的关键词。它主要包括两个功能: 首先是从匹配搜索查询的大量文本中选取一小部分句子, 也就是我们通常所称的片段 (Snippet); 然后从文本上下文中提取特殊的单词, 用特殊的彩色背景来标识它们。这样, 用户就可以很快地将注意力集中到这些要点上, 提高了阅读效率, 也增强了用户的搜索体验。如前所述, 为了使用该功能, 需要字段开启词条向量。

### (4) 空间搜索

在过去的几年中, Web 搜索从找寻基本的网页转换为找寻某些垂直领域的特定结果。其中很热门的一项就是地域的查询, 例如查询离我最近的餐厅、影院和理发店等。Lucene 的扩展包已经可以支持这个需求了, 使得用户可以通过提交基于地域的信息来对数据对象进行搜索。空间搜索面临的最大挑战就是, 对于每次查询而言, 用户的起点都会不同。因此必须在索引和查询期间使用空间逻辑来动态处理。Lucene 在这些方面已经做了不少性能上的调优。

进入 2015 年, Lucene 的最新版本已经更新到 5.4, 主要在易用性、维护操作、分布式



集群等方面进行了改进，并且进行了架构解耦，让索引结构可定制化和透明化，并向搜索框架方向发展。总体而言，Lucene 是相当不错的搜索引擎核心库。不过对于成熟的商业应用而言，其功能比较有限。有许多著名的开源项目都是基于 Lucene 实现的，它们新增了更为丰富的搜索引擎功能，例如聚合（Grouping or Aggregation）、切面（Facet，Lucene 4.0 之后也开始提供此项功能）、索引复制（Replication）和分片（Sharding）等。Solr、Elasticsearch、Hibernate Search、LinkedIn 的 Zoie 都是其中的成功案例。下面来介绍近几年比较流行的 Solr 和 Elasticsearch。

## 5.5.6 Solr 简介

Solr (<http://lucene.apache.org/solr/>) 是一个高性能的、基于 Lucene 的全文搜索服务器。它提供了比 Lucene 更为丰富的查询语言，实现了文档集合可配置，架构可扩展，并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。从 Solr 4.0 开始，其版本和它所集成的 Lucene 版本是绑定的，目前 Solr 最新的版本也是 5.4。Solr 继承了 Lucene 的很多要素，包括文档和字段的概念、相关性模型、各种模式的查询等。这里主要说下 Solr 的增强部分。

首先，增加了 RESTFUL (Representational State Transfer) 接口。RESTFUL 是一种软件架构的风格，提供了一组设计原则和约束条件，用于客户端和服务端交互类的软件。基于这种风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。例如下面的递进式链接就非常容易理解：

Get <a href="http://localhost:8983/solr/collection1/select/">http://localhost:8983/solr/collection1/select/</a>	在集合 1 中查询
Get <a href="http://localhost:8983/solr/collection1/update/">http://localhost:8983/solr/collection1/update/</a>	在集合 1 中更新
Get <a href="http://localhost:8983/solr/collection1/replication/">http://localhost:8983/solr/collection1/replication/</a>	复制集合 1 的索引数据

其次，可以通过配置文件，方便快捷地设置搜索引擎。Lucene 虽然提供了索引和查询的功能，但是几乎都是需要编码才能完成的，对专业知识的要求过高。那么能不能仅通过编辑文本，就能达到类似的效果呢？Solr 提供了这种可能。下面是其中最基础的 3 个文件。

□ schema.xml：用于定义某种文档类型的所有字段，及其类型、分析方式、存储选项等，包括后面提及的动态字段和复制字段。下面是一段示例代码：

```
<field name="id" type="string" indexed="true" stored="true" required="true" />
<field name="title" type="chinese_ik" indexed="true" stored="true" />
<field name="picture" type="string" indexed="false" stored="true" />
<field name="price" type="float" indexed="true" stored="true" />
<field name="type" type="int" indexed="true" stored="false" />
```

代码中的 chinese\_ik 是根据开源中文分词包 IKAnalyzer 自定义的分析器。schema.xml 里还可以定义各种语言的分析器和自定义的相似度<sup>①</sup>。Solr 会自动将这些转化为 Lucene 里对

① 这里的相似度就是用于相关性打分，原因是 Lucene 4.0 之前默认采用 VSM 模型的相似度作为相关性衡量，使得这一叫法沿用至今。

应的程序代码。

□ solrconfig.xml：定义索引和查询里常用的参数，以及各种自定义的 RESTFUL 风格的接口。例如索引阶段如何提交更新、数据写入哪些目录；查询阶段搜索哪些字段、缓存设置为多大、布尔默认操作符等。

□ solr.xml：定义管理、日志、分片和 Solr Cloud 的分布式集群。solr.xml 和 solrconfig.xml 的很多设置都已经超出了 Lucene 的能力范围，都是针对 Solr 增强功能的设置。

第三个增强的功能是支持动态字段 (Dynamic Field) 和复制字段 (Copy Field)。

动态字段是指在 Solr 的 schema 中没有固定名称的字段，它的名称由若干通配符来表示。在索引文档时，一个字段如果在 schema 的常规字段中没有被定义，它将和动态字段进行匹配。如果匹配成功，那么这个字段将使用该动态字段的定义。动态字段可以让系统更灵活，通用性更强。例如，一个动态字段设定为：

```
<dynamicField name="*_name" type="string" indexed="true" store="true" />
```

那么我们可以在索引时生成很多 Chinese\_name、English\_name、Russian\_name 这样的字段，而且它们都是统一采用 type="string" indexed="true" store="true" 的设定。

Solr 的字段复制机制，可以将多个不同类型的字段集中到一个字段。复制主要涉及两个概念，source 和 destination，一个是要复制的字段，另一个是要复制到哪个字段。比如：

```
<dynamicField name="*_name" type="string" indexed="true" store="true" />
<field name="names" type="string" indexed="true" store="false" />
<copyField source="*_name" dest="names" />
```

注意斜体加粗的部分，这会将各种语言的名字，统统都复制到 names 这个字段中。

第四个增强的功能为多租户 (Multi-tenancy)。基本概念是实现在同一个 Solr 服务实例上，可同时运行多个索引的建立和查询操作。Solr 是利用不同的文档集合 (Collection) 来实现的，配置里称之为核心 (Core)。一个核心代表一个独立的文档集合，这些独立的文档集合都有自己独立的 schema.xml 和 solrconfig.xml 配置。因为这些核心都在同一个 Solr 实例中，因此硬件资源是可共享的。

第五是增加了 DIH (Data Import Handler) 导入模块。这是 Solr 的一项特色，DIH 扩展可以将多种外在的数据源直接导入到 Solr 然后进行索引。只要是提供了 JDBC (Java DataBase Connectivity) 的数据，都可以和 DIH 配合工作，例如 Oracle、MySQL、微软的 SQL Server 等。你只需要提供数据库连接的参数，还有 SQL 查询语句，DIH 就能自动查询数据库，然后将结果集合转为 Solr 中的文档来索引。

最后，Solr 相对于 Lucene，甚至是 Elasticsearch 而言，最强大的地方在于有可视化的界面用于管理和监控整个集群的运行状况。图 5-7 是管理界面的主页示意图。其中你可以查看整体概况、日志报错、SolrCloud 集群状态、管理多租户 (多核心)、缓存配置等。这些对于初学者来说是非常人性化和直观的，同时也使得 Solr 上手的门槛更低。

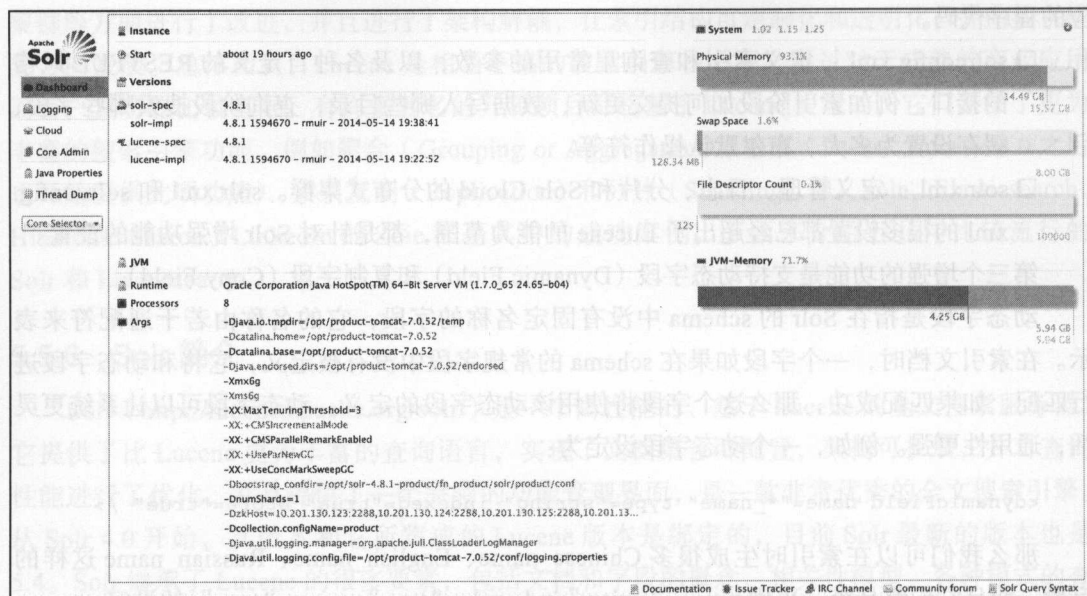


图 5-7 Solr 主界面的 Dashboard

当然，这些还不是全部。除了上述的要点，Solr 还有几大重点功能在实际应用中经常会采用，下面分别详细介绍：切面（Facet）和聚合（Grouping），以及分布式架构。

## 1. 切面（Facet）和聚合（Grouping）

针对传统的数据库和其他 NoSQL 数据存储进行比较时，你会发现切面是 Solr 中一个强大的功能<sup>①</sup>。让我们先来看一个生活中的例子。从图 5-8 可以看出，当你在一个购物网站上搜索“男士”时，你希望看到更多的选项来过滤结果。左侧方框标出的更多细分男士用品的分类，包括面部护肤品、服装配饰品、男士服装等。而右上侧的方框标出的则是更多细分的导购属性，如品牌、尺码、价格等。然后你就能选择分类，或者是选择属性，进一步缩小商品的范围了。例如选择“剃须刀”，返回的商品就会发生变化，而且相应的导购属性也会随之发生变化，如图 5-9 所示。这些都是切面产生的神奇功效。切面的搜索，你可以理解为“多个方面的浏览”，它允许用户在搜索集合上，看到一个更高层次的过滤条件和相应的统计，然后用户可以选择过滤条件，进一步缩小搜索的范围。

结果的聚合（Grouping）是 Solr 中另一个很有价值的功能，它可以保证为用户的查询返回最佳的结果组合。其逻辑是根据一个值<sup>②</sup>将结果进行分组。图 5-10 就展示了一个非常经典的应用场景。

① Lucene 在版本 4.0 之后也开始支持切面功能。

② 通常是某个字段值，也可以是复合的查询条件。





图 5-8 搜索“男士”的结果



图 5-9 在“男士”的搜索结果中, 选择电动剃须刀分类后的效果



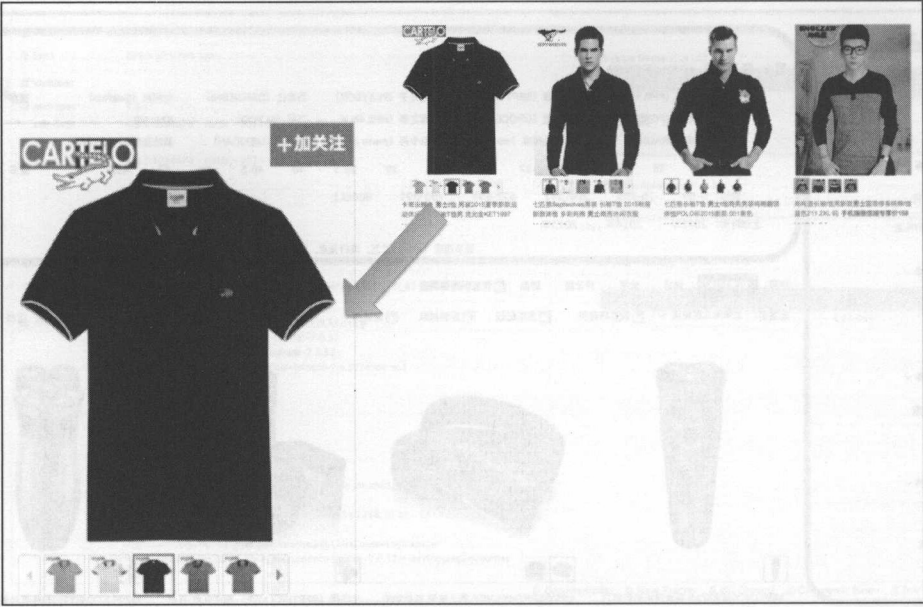


图 5-10 将同款中不同颜色的服饰聚合在 1 个展示位

我们在电商网站挑选服饰类商品时，常常可以在列表页就能预览到不同的颜色。实际上，同一款服装，在后台的索引数据中是通过多个文档来表示的。例如图 5-10 中的 T 恤衫，一共有 5 种颜色，5 个库存量，那么文档对象也是 5 个。如果直接搜索返回，就会发现同一件衣服占据了 5 个展示位，不仅浪费了黄金般的展示机会，也让顾客挑选无从下手，用户体验很糟糕。这个时候，按照唯一的款式码字段做一次聚合，就能将同样一款服装、不同的颜色归并到同一组，前端展示就非常容易了。大致的聚合结果类似于：

```
"男装" - 100 件
    "男装 A" - 11 件
        "酒红色" - 4 件
        "海蓝色" - 4 件
        "亮白色" - 3 件
    "男装 B" - 20 件
    ...
```

目前 Solr 还不支持多层级的聚合，大的聚合中无法再做小的聚合。另外一点需要注意的是，聚合和去重功能有所不同。聚合会保留同一组的结果集合（通常也不是完全相同的），而去重则会完全一样的重复数据去除掉，不再返回冗余文档，以保证唯一性。当然，你也可以通过自定义将聚合用于去重。

读者可能已经发现，切面和聚合是非常类似的。切面的查询结果主要是分组信息：有哪些分组，每个分组包括多少个文档；但是分组中有哪些具体的数据是不可知道的，只有进行进一步的搜索。这也保证了切面的查询速度是相当快的，对性能的消耗也很小。聚合则类

似于关系数据库的 group by，可以用于一个或几个字段的去重，或者显示一个分组的若干条记录等。不过这也导致了聚合操作相当耗费性能，因此要结合业务场景来选择，慎重使用。

## 2. 分布式架构

Solr 的命名来自 Search On Lucene w/ Replication，可见其非常注重系统的容错性和扩展性，这也是 Lucene 所不具备的。因此，我们一定要阐述一下 Solr 分布式架构的理念。先来看分布式系统里最基本的两个概念：分片和副本。

❑ 分片 (Sharding)：当有大量的文档时，由于内存和硬盘处理能力的限制，单台机器可能无法快速地响应客户端的请求。在这种情况下，数据可以被切分为较小的部分，称之为分片。在 Solr 系统中，每个分片都是可以独立运作的 Lucene 索引。每个分片可以放在不同的服务器上，并可以在集群中传播。当需要查询的索引分布在多个不同的分片上时，分布式系统会将查询分发给每个相关的分片，并将结果合并在一起。这些对上层应用而言都是透明的，它们并不知道底层发生了什么。

❑ 副本 (Replication)：为了提高吞吐量或实现高可用性，可以使用副本。副本是一个数据集的精确复制，通常和分片结合起来使用。因此，每个分片都可以有多个副本，万一有机器出现故障，上面的若干分片无法提供服务时，集群会主动查找其他正常机器上该分片的副本。

在利用分片和副本的概念搭建的分布式环境下，索引和查询的过程会有所变化。索引阶段增加了一个更新传播的过程。当发送一个新的文档给集群时，接收到变化的机器 A 就会知道该文档应该被放入哪个分片，而且该分片分散在哪些其他的机器上。这样更新的文档就可以分发到合适的机器上（也可能包括 A 自己），从而对分片进行更新。图 5-11 是该过程的示意图。

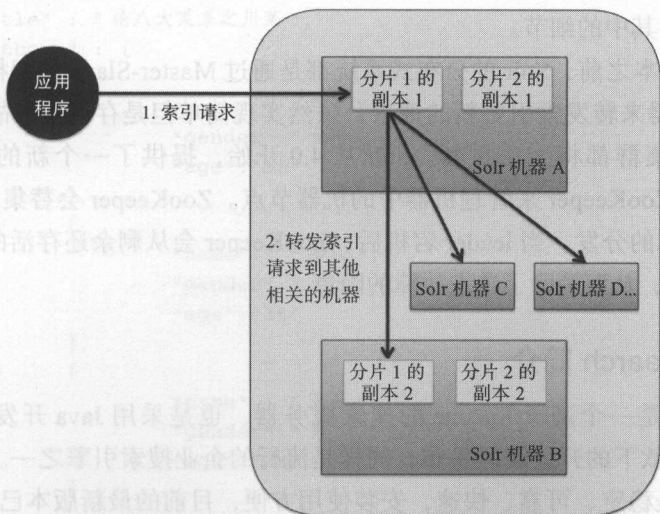


图 5-11 分布式环境中分发索引的更新请求

查询阶段，增加了一个结果合并的过程。收到查询请求的机器 A，会将查询转发给保存了指定索引分片的所有其他机器，要求它们（也可能包括 A 自己）进行查询并返回相应的结果。收到所有返回后，机器 A 对它们进行合并，包括去重、排序等，然后将最终结果返回给客户。图 5-12 是该过程的示意图。

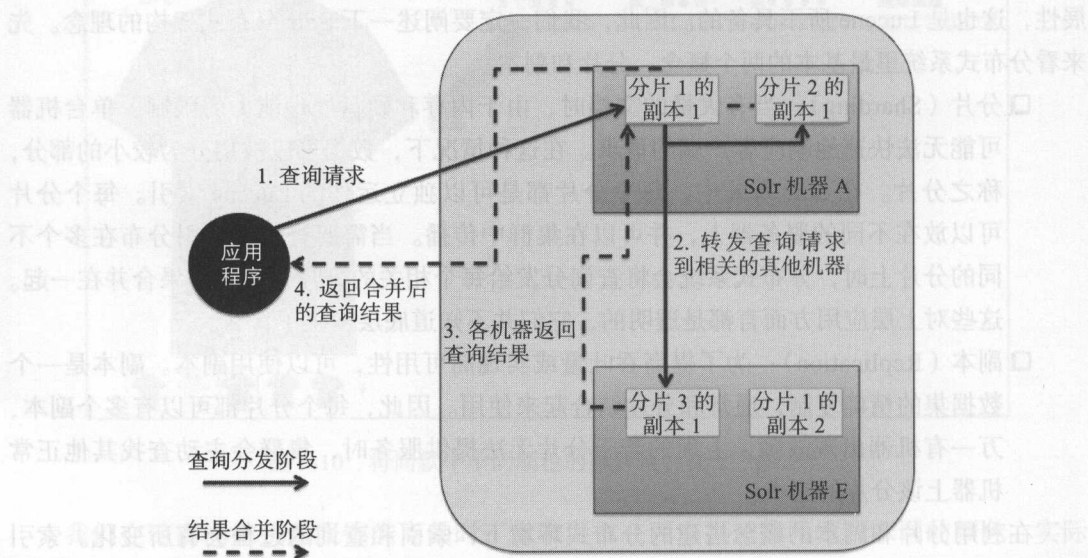


图 5-12 分布式环境中分发查询的请求，并合并查询结果

可能有读者会说，分布式环境导致索引和查询都变得更加复杂了。确实如此，不过，好在 Solr 这样的分布式系统都替你实现好了，对应用开发者而言这些都是透明不可见的，大家通常不用关心其中的细节。

在 Solr 4.0 版本之前，Solr 的分布式系统都是通过 Master-Slave 的架构来实现的，由专门的 Master 服务器来转发索引更新的请求，虽然实现简单但是存在单点故障，万一 Master 宕机，那么整个集群都将无法更新。Solr 从 4.0 开始，提供了一个新的分布式架构——SolrCloud，利用 ZooKeeper 来管理机器中的机器节点。ZooKeeper 会替集群选出一位 leader 角色进行更新请求的分发。当 leader 宕机后，ZooKeeper 会从剩余还存活的机器中再次自动选举出新的 leader，从而消除了单点故障的隐患。

### 5.5.7 Elasticsearch 简介

Elasticsearch 是一个基于 Lucene 的搜索服务器，也是采用 Java 开发的，它的源码作为 Apache 许可条款下的开放源码发布，同样是流行的企业搜索引擎之一。设计目标是达到实时搜索，优点是稳定、可靠、快速，安装使用方便，目前的最新版本已经更新到 2.1。和 Solr 一样，Elasticsearch 也是基于 Lucene 架构的，因此很多要素都是一脉相承的，例如文

档和字段的概念、相关性的模型、各种模式的查询等。相对于 Lucene 而言, Elasticsearch 增加了 RESTFUL 风格的接口, 用 JSON 格式传输数据, 支持动态映射, 是一个分布式的、用租户模式的搜索引擎。

□ RESTFUL (Representational State Transfer) 接口: 和 Solr 类似, Elasticsearch 允许用户通过 RESTFUL 风格的链接对其搜索引擎进行操作。例如下面的递进式链接就非常容易理解。

Get <http://localhost:9000/> 获取 Elasticsearch 的基本信息  
 Get [http://localhost:9200/\\_cluster/state/nodes/](http://localhost:9200/_cluster/state/nodes/) 获取集群中节点的信息  
 Get [http://localhost:9200/\\_cluster/nodes/\\_shutdown](http://localhost:9200/_cluster/nodes/_shutdown) 向集群中的所有节点发送“关闭”的命令。

□ JSON (JavaScript Object Notation) 格式: 不像 Solr 能支持较多的格式, Elasticsearch 基本上是以 JSON 为数据传输的格式, 其他格式需要额外插件的支持。JSON 是一种轻量级的数据交换格式, 完全独立于具体的计算机语言, 能够跨平台使用。这些特性使得 JSON 易于被人们阅读和编写, 同时也易于被机器高速地解析、生成和传输, 因此 JSON 成为理想的数据交换语言。例如, 一篇美食的文章就可以长成这样:

```
{
  "id" : "10012",
  "title" : "论八大菜系之川菜",
  "authors" : ["张三", "李四", "王五"],
  "date" : "2015.08.20"
}
```

JSON 还能够支持层次型的数据结构, 例如:

```
{
  "id" : "10012",
  "title" : "论八大菜系之川菜",
  "authors" : [
    {
      "name" : "张三",
      "gender": "女",
      "age": "36"
    },
    {
      "name" : "李四",
      "gender": "男",
      "age": "35"
    },
    {
      "name" : "王五",
      "gender": "男",
      "age": "28"
    }
  ]
}
```



```

"date" : "2015.08.20"
}

```

相应的，Elasticsearch 也提供了对多层次嵌套型数据索引的支持，这是基础 Lucene 所不具备的。

- ❑ 映射 (Mapping)：类似 Solr 的 schema，用于定义不同字段的基础类型、索引分析、存储选项及其他相关设定。值得一提的是，Elasticsearch 支持动态的匹配，也就是说无须事先确定映射，而是让系统根据索引对实际输入的文档直接进行判定，以确定每个字段的类型。当然，自动判断不可能每次都完全符合用户的预期，最稳妥的方式还是事先定义。
- ❑ 分布式 (Distributed)：与 Solr 类似，也有分片 (Sharding) 和副本 (Replication) 的概念，分布式索引和查询的原理也基本相同。需要注意的一个细节是，Elasticsearch 的副本是不包含主分片的。如果有 1 个副本，实际上意味着有两个分片可以提供搜索，1 个主分片加上 1 个副本。同理，如果有 3 个副本，实际上有 4 个分片可供搜索。
- ❑ 多租户 (Multi-tenancy)：可以根据不同的用途来区分索引，同时操作它们，并保证资源的分配和隔离。不像 Solr 需要 schema.xml、solrconfig.xml 等这种高级设置，Elasticsearch 的实现比较简洁和直观。

综上所述可以看出，Elasticsearch 和 Solr 有很多共同点，同时也有不尽相同之处<sup>①</sup>。

Solr 的相对优势：

- ❑ 有可视化的界面，使用者可以清晰地看到集群的状态、统计数据、索引大小等基本信息。
- ❑ 文档和字段配置步骤清晰，按照教程很容易上手。
- ❑ 除了 JSON，还支持 XML 和 CSV 格式。
- ❑ 即使索引生成后，其分片也可以添加或再次切分，是一个更为灵活的分布式配置方案。

Elasticsearch 的相对优势：

- ❑ 除了传统的 SQL 数据库和文本，还支持一些非 SQL 的大数据解决方案，例如 MongoDB 和 Redis 等；也支持更实时性的数据源导入，例如 Kafa、ActiveMQ 等这种消息队列。
- ❑ 当实时性更新操作非常频繁的时候，读取和查询的效率仍然可以保持较高的水准。
- ❑ 除了普通的扁平结构文档之外，还支持层级型文档的索引。
- ❑ 可以使用脚本语言，来自定义排序时的提升 (Boosting) 功能。
- ❑ 无需 Zookeeper 来配置分布式集群。完全依靠自身的节点发现、加入和恢复功能。
- ❑ 聚合 (Elasticsearch 称之为 Aggregation，而 Solr 称之为 Grouping) 支持嵌套的层级，大组可以包含更细力度的分组。理论上是支持无限次嵌套，但是实际中需要考虑到查询的性能和效率。例如：大组按照衣服的颜色来聚合，小组按照尺码来聚合。这

<sup>①</sup> 请注意这里的“不同”或“优劣势”都只是暂时的，随着双方版本的不断演化情形可能会发生改变。

样用户可以先选择颜色,再选择尺码。大致的层次结构如下:

"女装" - 100 件

"女装 A" - 11 件

"酒红色" - 4 件

"尺码 XS"

"尺码 S"

"尺码 M"

"尺码 L"

"海蓝色" - 4 件

"尺码 XS"

"尺码 S"

"尺码 M"

"尺码 L"

"亮白色" - 3 件

"尺码 S"

"尺码 M"

"尺码 L"

"女装 B" - 20 件

"天蓝色" - 5 件

...

...

□ 时光之门 (Gateway)。“时光之门? 好像星际争霸里神族的兵营啊!”大宝一阵兴奋。

“哈哈,类似科幻巨作里的时光之门是不是很酷呢?有了它,我们能在过去、现在和未来穿梭自如,能挽回很多缺憾之事。”小明接着说道。Elasticsearch 同样提供了这个神奇的魔法,为你的集群索引和配置元数据提供了数据镜像服务。一旦整个集群崩溃,或者是因为特殊需要而进行关停,我们就可以让集群恢复到最后的一个状态,并且让服务重新启动。

总体而言,Solr 上手较快,配置步骤较严谨,更适合入门者。而 Elasticsearch 的配置较灵活,实时更新和查询性能更好,更适合有一定经验,且拥有超大规模数据的用户。当然,这些都是基于现状的,相信随着两个开源项目的不断完善和相互借鉴,我们将会看到更为成熟的搜索引擎实现方案。

“太感谢小明哥了,我对搜索引擎已有所领悟,还介绍了这么好的开源项目,回去马上练练手!”

“哈哈,看来你对新技术很有热情嘛!别急,介绍完搜索引擎,让我们来看看推荐系统和在线广告。严格来说,这两者不仅仅是普通的检索系统。相对于搜索引擎,它们需要更多的数据挖掘、竞价拍卖等技术的支持,是综合性的信息系统。不过,对于普通用户而言,其基本的形式还是根据用户某种形式的‘输入’,返回用户可能感兴趣的某类‘信息’,因此我们还是将其归为信息检索这个大类。”

## 5.6 推荐系统

广义上来讲，推荐是一种为用户提供建议，帮助其挑选物品并做出最终决策的技术。例如，为用户展示热销商品的排行榜就是一种推荐。当然，推荐热门物品的技术难度不高，用户转化率也不一定很理想，所以这里将探讨个性化的推荐（这里并不意味着个性化是推荐系统的专利，搜索引擎同样很需要个性化，在 5.5 节也有所提及）。个性化推荐系统是建立在海量数据挖掘基础上的一种高级信息检索平台，它会根据用户所处的情景，以及用户的兴趣特点，向用户推荐可能感兴趣的信息和商品。搜索和推荐引擎天生就是一对孪生兄弟。之所以是兄弟，那是因为它们都是人们查找信息的工具。这点就决定了这两者所需处理的数据，以及返回给用户的信息往往都是同质的。但是它们也有很明显的不同点：

- ❑ 传统的搜索利用的是集体行为，而推荐则是挖掘个人行为。所谓集体行为，就是在命中关键词后，搜索会看看大部分的用户关心的是什么信息，最后返回给用户。而推荐则是直接查看当前用户的历史行为和所处的情景，猜测他/她最关心的是什么信息。
- ❑ 搜索的输入是明确的关键词，而推荐往往没有明确的查询条件。搜索引擎越来越为我们所熟悉，往查询框里输入若干关键词是必不可少的一步。而推荐只要有这个用户之前的历史行为数据，包括查询、浏览、购买等信息，根据算法分析就可有大致推测，查询关键词并不是必须的。

列举一个更形象的例子，当用户输入关键词“中国美食”，那么搜索引擎会返回时下关于美食的各种热门话题，包括各大菜系历史文化、各地著名的餐馆、美食主题的狂欢节等。而推荐引擎则会根据这个用户之前的喜好，猜测他/她更偏爱对历史文化的研究，可能在其还没有下达任何查询指令的时候，就已经推荐给他/她更多文化相关的文章。搜索应对的是用户明确的信息需求，需要高效率的查询性能（当然，明确也是相对而言的。如果用户仅仅输入“苹果”，恐怕连人类都很难知道他/她是需要营养的水果呢，还是高端的电子产品）。而推荐无需用户的明确需求，因此更注重的是算法的精准性。

正因为推荐具有无须用户主动输入的特色，所以它对搜索行为可以起到很好的补充作用，主要包括：

- ❑ 增加物品被浏览、被销售的数量。用户在没有任何查询的情况下，同样可以看到他/她可能感兴趣的物品，这个更符合线下“逛街”的感觉，更容易增加向上销售（Up Sell）和关联销售（Cross Sell），让用户买得更多。
- ❑ 出售多样化的商品。搜索通常都会返回畅销品，导致小众的、新颖的长尾（Long-tail）物品露出不够。但这并不代表某个顾客不喜欢这类物品。推荐能够根据用户的喜好，向他们提供发现新奇的机会。
- ❑ 增加用户的满意度和忠诚度。相对于搜索，良好的推荐更容易让用户觉得系统更“懂”人心，长期使用自然会增加用户的好感和黏度。如果持续针对这些用户行为进



行数据挖掘,并进一步优化推荐系统,那么用户的满意度会明显提升,这样也能缓解互联网站点普遍存在的用户忠诚度较低的问题。

### 5.6.1 推荐的核心要素

推荐引擎的系统 and 算法发展至今,也有二十多年的历史,各种方法层出不穷。为了让大家更好地理解主流趋势,先来归纳一下推荐的三大要素。

#### 1. 系统角色

抽象来看,推荐系统中一般有四个重要的角色:用户、物品、情景和匹配引擎。用户是系统的使用者,物品就是将要被推荐的候选对象,情景是推荐时所处的环境,而引擎就是用于匹配用户和物品的核心技术。例如,亚马逊网站的顾客就是用户,网站所销售的商品就是物品,浏览的地理位置和时间就是情景,而研发团队提供的关键算法就是匹配引擎。因此,推荐系统可以认为是在一定的情景下,比较用户的信息需求和物品特征信息,使用相应的匹配算法进行计算筛选,最终给用户推荐其可能感兴趣的物品。最后,值得注意的是,这里的用户角色都是现实中的自然人,同时,某些场景下被推荐的物品角色可能也是现实中的自然人。例如,一个招聘网站会给企业雇主推荐合适的人才,这时候应聘者担当的就是物品角色。如果向应聘者推荐合适的企业雇主,那么雇主担当的就是物品的角色。针对这种特殊情况我们不会做单独说明,这里并非说将人或企业当作物品来买卖,而只是为了区分推荐系统中的不同角色,以便于后面的解释。

#### 2. 相似度

推荐一般是基于这样两个假设:

- 假设用户对物品 a 感兴趣,那么和 a 相似的物品 b、c、d 也会引起他/她的兴趣。
- 假设用户 B 和用户 A 相似,那么 B 感兴趣的物品也会引起用户 A 的兴趣。

因此,推荐在很大程度上要关注如何衡量物品之间的相似度,以及用户之间的相似度。你可能会问这里的“相似度”和搜索引擎的“相关性”有什么区别?主要是应用场景不同。搜索里是将用户输入的条件和待查询数据相匹配,两者是不对等的,因此业界称为相关性;推荐里没有用户的主动输入,而是通过研究物品和物品之间、用户和用户之间存在多少相似的特征,来达到建议的目的。相比较的对象都是对等的,因此业界称之为相似度。从技术实现的角度来理解,相似度和相关性是互通的,因此相似度同样可以利用向量空间模型 VSM、概率模型等来刻画。

#### 3. 相似度传播框架

现实生活中的推荐,常常来源于“口口相传”,上述的两个假设体现了这点同样适用于线上。而且我们可以利用相似度的传播性,帮助用户进一步发现更多潜在的兴趣。例如,如果物品 a 和 b 相似, b 和 c 相似,那么 a 和 c 也可能存在一定的相似度。



5.6.2 推荐系统的分类

了解了核心要素后，我们就可以根据这些来对推荐系统进行划分了。

首先，按照推荐依据来划分，可分为如下三类。

❑ 基于物品：给定物品 a 后，按照其他物品和 a 相似度的高低来推荐。典型的应用场景就是在浏览商品的详情页时，左侧的“看了此商品还看了”、“买了此商品还买了”等推荐栏位，如图 5-13 中左框标出的列表，推荐了和当前苹果类似的其他水果。



图 5-13 基于物品推荐的示例，针对某款苹果的左侧推荐栏位

❑ 基于用户：给定用户 A 后，按照其历史行为所构建的用户模型来推荐。典型的应用场景就是个性化首页中的“猜你喜欢”模块，如图 5-14 所示，此位顾客一定是位时尚达人。

❑ 基于情景 (Scenario)：情景也可以翻译为场景、情境，业界还有人称之为上下文 (Context)，其本身并没有严格的定义，简单点说就是指用户所处的信息环境。用户浏览的网页、所处的地理位置、当时的季节和气温等，都可以算在这个范畴之内。在很多推荐应用中，仅仅只考虑用户和物品很可能是不够的。在某些特定场景下，

将环境信息整合到推荐流程也是很有必要的,例如对于度假旅行的线路,夏季建议承德避暑山庄是很棒的主意,而冬季最好建议海南沙滩狂欢节。还有,中午饭点到了,在寻找餐厅的时候你当然希望就近解决,对于需要1个小时才能到达的地点你的肚子可能不会乐意。图5-15就会告诉你,在上海的天山路附近有哪些美食,排名前几位的离你的当前距离都没有超过700米。不难看出,移动端的应用更符合情景模式下的推荐。



图 5-14 基于用户推荐的示例,针对某位用户的喜好进行推荐

“那么这3种各有什么特点呢?”好奇的大宝再次发问。

“使用物品作为推荐的依据,需要较为完善的物品信息数据,例如标题、产地、颜色、口味等其他领域相关的属性。一旦拥有这些数据,不需要有用户访问物品的记录,也能进行推荐,比较适合用户访问量还不大的系统。使用用户作为依据时,可以不需要太多与物品相关的信息,但是需要累计用户的访问日志,需要一定的流量作为基础,否则会面临冷启动的问题(系统无法在量级非常有限的数据集上进行有效的计算和挖掘)。使用情景作为依据来推荐时,对物品和用户的数据要求都会降低,代价是需要额外收集用户所处的场景,例如地理位置,不仅需要获得用户的许可,而且还要进行实时的更新。”



图 5-15 基于场景的示例，针对当前地理位置的推荐

接着按照相似度的定义来划分，可分为如下四类。

- ❑ 基于内容：其关键是通过人工运营或自动抽取的特征进行推荐。以博客的文章为例，假设它是物品角色，那么它的内容特征可以包括文章的标题、文体、作者、时间等。而对于用户角色，内容特征一般是人口统计学信息等，比如年龄、性别、地区、职业、爱好等。因此，基于内容的推荐需要考虑是否有自动化的技术能帮助运营人员来便捷地获取这些特征，考虑如何维护并持续更新，以及如何通过这些数据进行相似度的计算。如果这些都能够实现，那么基于内容的方法就有着明显的优势：无需任何用户的访问行为，就可以仅仅根据内容的特征，进行基于物品或基于用户的推荐。此外，在特定的领域中，人工的标注会提供更有价值的线索，从而提升推荐的满意度。
- ❑ 基于知识：这种方式和基于内容的推荐比较相近，不过多了一些通过人类知识定义



的逻辑规则,因此需要人为地提供大量专业领域的知识,构建成体系的知识库,并和用户产生交互。根据用户交互的形式,又可以细分为基于约束的和基于实例的。两者的形式比较类似,都是让用户指定需求,然后推荐系统给出答案。如果找不到合理的,那么用户就需要再次修改需求。这个方式和搜索更为接近,需要用户较多的精力,可惜在互联网时代,用户都是偷懒的,很少有人愿意这么做。综合建立知识体系和用户参与的成本,这个方法一般仅限于学术研究。当然,它也有更为精准的优势。

□ 基于用户行为: 这种方法是通用户和物品之间的关系(区别于人与人之间的交互)进行推荐。物品和物品之间的相似度,可以不再通过内容特征来计算,而是通过用户访问来刻画。例如,物品 a 经常被用户 A、B、C 访问,而物品 b 同样也经常被 A、B、C 访问,那么我们就认为物品 a 和 b 之间也有相似度。最著名的协同过滤(Collaborative Filtering)就是这方面的典型案例。不过需要注意的是,协同过滤虽然最早是完全利用用户访问物品这种行为关系的,但时至今日,已经有很多其他方式也被整合到其中了,协同过滤更偏向于一种推荐的框架,后面在介绍相似度传播时会详细介绍。相对于基于内容和知识的方式,基于用户行为的前期运营成本则很低,甚至是只要累积用户流量就能达到推荐的目的,不过精准度往往不如前两者高。

□ 基于社交和社区: 这种方式是通过用户之间的关系来进行推荐的。最近几年,随着 Web 2.0 时代的到来,用户社交网络迅猛发展,大家可以建立朋友圈,发表观点,相互评论和点赞。这些都促使推荐系统诞生了新兴的方式,比如根据人与人之间的交互行为,判断用户之间的相似程度等。其优势在于,如果用户之间存在相似度,那么可信度就会较高,推荐效果也会更为理想。不过,对于数据源的要求就实在有些高,很多时候我们也无法保证推荐系统能获得用户的社交信息。

下面是按照相似度传播<sup>①</sup>的方式来划分。

□ 无传播: 通常是基于内容和知识的推荐,都没有考虑用户对物品的访问行为。

□ 协同过滤(Collaborative Filtering): 协同过滤是基于最直观的“口口相传”来传播的,假设我们愿意接受他人的建议,尤其是很多人都向你建议的时候。其主要思路就是利用已有用户群过去的行为或意见,预测当前用户最可能喜欢哪些东西。根据推荐依据和传播的路径,又可以进一步细分为基于用户的过滤和基于物品的过滤。

基于用户的协同过滤是指给定一个用户访问物品的数据集合(假设访问就表示有兴趣),找出和当前用户历史行为有相似偏好的其他用户,将这些用户组成“近邻”。然后对于当前用户没有访问过的物品,利用其近邻的访问记录来预测。根据访问关系图 5-16 来看,用户 A 访问了物品 a 和 c,用户 B 访问了物品 b,用户 C 访问了物品 a、c 和 d。我们可以计算出用户 C 是 A 的近邻,而 B 则不是。因此系统会向 A 推荐 C 访问的物品 d。

① 这里的传播指的是相似度通过物品和用户两种角色之间的交互关系进行扩散。

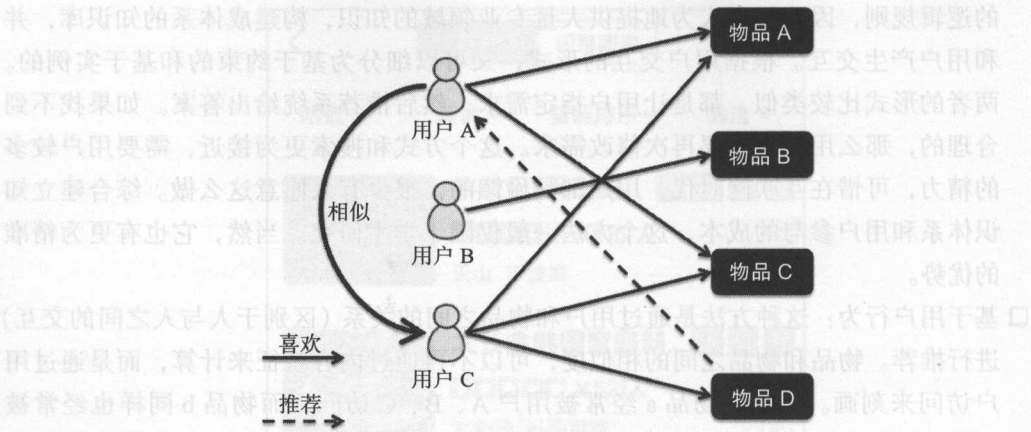


图 5-16 基于用户的协同过滤原理

基于物品的协同过滤是指利用物品的相似度，而不是用户间的相似度来计算预测值。在图 5-17 中，物品 a 和 c 因为都被用户 A 和 B 同时访问，因此它们被认为相似度更高。当用户 C 访问过物品 a 后，物品 c 也会被推荐给他 / 她。

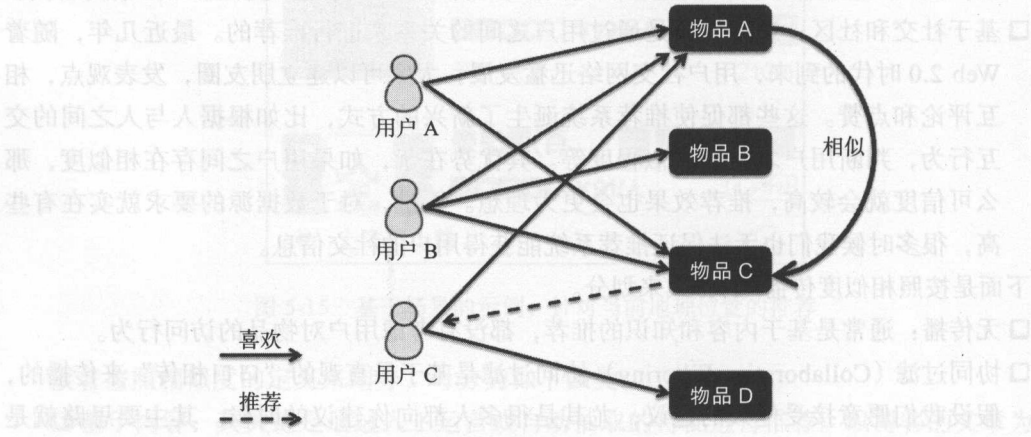


图 5-17 基于物品的协同过滤原理

“看上去，基于物品和基于用户的方式差不多嘛，只是观察数据先后顺序不同而已啊！”大宝产生了疑惑。

小明：“别小看这个差异哦，由于传播的方向不同，基于用户和基于物品也会有一些区别，下面就来介绍一下它们的区别。”

- 准确性：推荐系统的准确性在很大程度上依赖于系统中用户数和物品数之间的比例。通常情况下，一小部分相似度高的用户，其价值远远高于一大部分相似度较低的近邻。对于用户数量远远大于物品数量的大型商业系统（例如一个 B2C 的购物网站）

而言,如果用户之间的区分度不够,那就很难界定哪些是真正高相似度的用户,因此采用基于物品的协同过滤更为精准。同理,对于物品数量远远大于用户数量的系统(例如内部文献系统)而言,采用基于用户的协同过滤则更为精准。

□ 高效性:虽然数据挖掘的部分是离线计算,并不要求实时返回结果,但我们也不希望其消耗的时间过于离谱,更何况,目前有些应用已经需要实时性的挖掘结果了。当用户数量远远大于物品数量时,物品的相似度计算所消耗的资源要远远小于用户的相似度计算,因此基于物品的协同过滤效率更高。反之,基于用户的协同过滤会更高效。

□ 稳定性:物品和用户总是在不断变化。变化就意味着用户和物品之间的关系需要更新,协同过滤的结果也需要做出相应的改变。如果系统中物品的集合比用户的集合更稳定,那么基于物品的方法会避免频繁地数据计算和更新,因此更实用一些。反之,基于用户的方法更适用于用户集合相对稳定的系统。

介绍完了基于用户和物品的协同过滤,现在来看看基于聚类的协同过滤和多次协同过滤。

基于聚类的协同过滤是基于用户的协同过滤的变种,其中用户的角色被替换为一组具有类似兴趣的用户,可以认为上述的“近邻”就是一种用户集合。这样会使得相似度传播的范围更为广泛,但是精确度会有所下降。

至于多次协同过滤,如果基于聚类的过滤是将用户节点进行合并的话,那么多次协同过滤就是在用户和物品之间添加更多的访问连线。例如,在第一次协同过滤计算完毕后,我们会将物品 d 推荐给用户 A,那么在第二次过滤中,我们就可以假设 A 就是喜欢 d 的,并将此作为既成事实的数据加入计算。同理,这也会通过牺牲精确度来换取更多的推荐结果。

“从无传播,到协同过滤,再到更多层级的相似度传播,我们可以看到精准性会越来越差,但是新颖度会越来越高,可能会给用户带来一定的惊喜,在实际应用中这是需要仔细权衡的。”

“谢谢小明哥的经验之谈!”

### 5.6.3 混合模型

看了如此多的推荐分类,它们在不同的应用领域表现出的效果各有千秋,也各有优劣。因此,业界也会考虑构造一种混合的体系,结合不同算法和模型的优点,尽量克服单个算法和模型所面临的缺陷和问题。混合的方式大体上可以分为微观混合和宏观混合。

□ 微观混合:将不同的特征混合起来使用,例如将基于内容和基于用户行为的相似度计算结合起来,这样基于内容的方式也可以加入协同过滤的传播框架,解决其面临的冷启动问题。或者是将用户的社交信息加入用户近邻的选择,提高协同过滤推荐的可信程度。

□ 宏观混合:相对于微观混合,宏观的方式不关心特征的合并,而是注重将不同推荐系统的结果有机地结合起来。只要是能推送结果的系统,都可以加入进来,因此更



为灵活。例如，我们可以让基于用户、基于物品和基于情景的三个系统同时工作，然后根据合并、加权、轮播等方式进行混合。

#### 5.6.4 系统架构

综合上述内容来看，推荐系统的主要模块分为数据收集、用户建模、物品建模、推荐算法、混合模块、结果存储、前端展示和查询引擎，其中大部分是离线的操作。

离线部分涉及如下内容。

- ❑ 用户建模：根据用户的人口统计学信息和用户行为数据，建立用户画像等模型，刻画其短期和中长期的兴趣。
- ❑ 物品建模：根据物品的领域属性，以及用户访问这些物品的数据，建立物品画像模型，刻画其本质特征。
- ❑ 推荐算法：根据用户和物品的建模，通过不同的推荐方式进行演算，最终找到能与用户或物品输入所匹配的推荐物品。
- ❑ 混合模块：根据不同的混合策略，将多种方式的推荐结果进行合并。因为考虑到实时性，一般都被放入离线处理。当然，如果系统足够轻量级，混合逻辑并不复杂，数据量也足够小，是可以放入在线部分来处理的。
- ❑ 结果存储：将推荐算法的挖掘结果保存下来，以便于在线的实时访问，倒排索引同样是一个不错的选择。当这些结果数据达到一定的规模，或者是包含了比较复杂的商业逻辑时，就可以考虑直接使用搜索引擎来协助了。

在线部分涉及如下内容。

- ❑ 数据收集：因为用户的行为会作为很多推荐算法的数据来源，因此需要通过 Flume 之类的框架来收集用户访问的日志。当然，用户使用搜索和推荐引擎本身的数据也会被记录，并以此来对之后的算法做进一步的优化。
- ❑ 前端展示：这一部分用于接收网页或移动设备发过来的推荐请求，并经过必要的初步处理之后向推荐后端引擎传递，并在拿到后端返回的结果之后返回给前端用户。
- ❑ 查询引擎：推荐系统的复杂逻辑基本上都是在离线时完成的，因此通常情况下在线查询只需要使用搜索这样的高效检索系统来完成就行了。

最后，整体的系统框架示意图如图 5-18 所示。

#### 5.6.5 Mahout

最近几年开源的推荐引擎或算法项目开始逐步崭露头角，这里介绍一个有代表性的项目——Apache 的 Mahout (<http://mahout.apache.org>)。Mahout 这个单词本身的意思是象的饲养者和驱赶者。Mahout 的分布式挖掘中会使用 Apache Hadoop 作为数据的存储，来实现更高的可伸缩性和容错性，而 Hadoop 的徽标上有一头黄色的大象，Mahout 希望能让这头大象发挥更好的作用，这大概就是“Mahout”这个名字的由来。

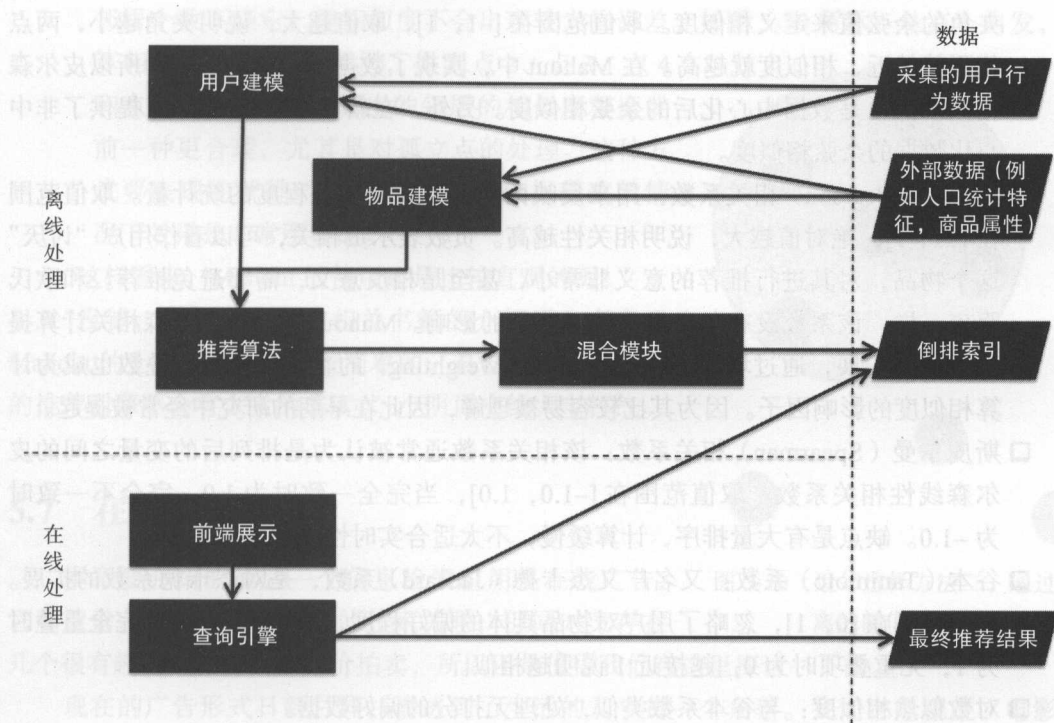


图 5-18 推荐引擎常见的系统架构

Mahout 项目和 Apache Lucene 也颇有渊源。Lucene 开源搜索社区中的一些成员，对机器学习非常感兴趣，因此发起了一个新的项目，他们希望实现一些常见的用于数据挖掘的机器学习算法，并拥有良好的可扩展性和维护性，以达到帮助开发人员方便快捷地创建智能应用程序的目的。该社区最初基于一篇关于在多核服务器上进行机器学习的学术文章进行原型的开发，此后在发展中又并入了更多更广泛的机器学习方法。因此，我们要特别注意，Mahout 并非只限于推荐算法，它还提供了分类、聚类和频繁项挖掘的算法。关于这些数据挖掘的内容将会在第 6 章中详细介绍。此外，通过使用 Apache Hadoop 库，Mahout 可以有效地扩展到云中。就推荐算法而言，Mahout 主要实现了协同过滤的方法，包括基于用户的推荐和基于物品的推荐。目前 Mahout 的版本更新到了 0.11 版。

在 Mahout 的推荐场景中，用户对物品的偏好形成了一个二维矩阵，并将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，或者将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度。Mahout 关于相似度计算的实现都是基于向量的距离的，距离越近相似度越大。下面我们来介绍几种常用的计算方法。

- 欧氏距离相似度：利用欧氏距离定义的相似度，取值范围在  $[0,1]$ ，其值越小，说明距离越近，相似度越高。
- 余弦相似度：和向量空间模型（VSM）类似，利用多维空间两点与所设定的点形成

夹角的余弦值来定义相似度。取值范围在  $[-1, 1]$ ，取值越大，说明夹角越小，两点相距就越近，相似度就越高。在 Mahout 中，实现了数据中心化的过程，所以皮尔森相似度值也是数据中心化后的余弦相似度。另外，在新版本中，Mahout 提供了非中心化数据的余弦相似度。

- ❑ 皮尔森 (Pearson) 相关系数：用来反映两个变量线性相关程度的统计量。取值范围在  $[-1, 1]$ ，绝对值越大，说明相关性越高。负数表示负相关，可以看作用户“讨厌”这个物品，对其进行推荐的意义非常小，甚至是相反意义，需要避免推荐。和欧氏距离一样，该系数没有考虑重叠数对结果的影响。Mahout 中，为皮尔森相关计算提供了一个扩展，通过增加一个枚举类型 (Weighting) 的参数来使得重叠数也成为计算相似度的影响因子。因为其比较容易被理解，因此在早期的研究中经常被提起。
- ❑ 斯皮尔曼 (Spearman) 相关系数：该相关系数通常被认为是排列后的变量之间的皮尔森线性相关系数。取值范围在  $[-1.0, 1.0]$ ，当完全一致时为 1.0，完全不一致时为 -1.0。缺点是有大量排序，计算缓慢，不太适合实时性强的推荐系统。
- ❑ 谷本 (Tanimoto) 系数：又名广义杰卡德 (Jaccard) 系数，是对杰卡德系数的扩展。取值范围在  $[0, 1]$ ，忽略了用户对物品具体的偏好程度，只考虑 0 或 1。完全重叠时为 1，无重叠项时为 0，越接近 1 说明越相似。
- ❑ 对数似然相似度：与谷本系数类似，处理无打分的偏好数据。

如果对这些方法具体的计算公式感兴趣，可以参见第 6 章关于数据挖掘的介绍。Mahout 对于相似近邻的选择也值得一提。上文提到过，协同过滤的推荐中，需要选择和当前用户兴趣或物品特性类似的近邻。近邻的计算对于推荐数据的生成是至关重要的，Mahout 划分邻居的方法有两类，具体如下。

- ❑ 固定数量的近邻：用“最近”的 K 个用户或物品作为邻居。如图 5-19 所示，假设要计算点 1 的 5 个邻居，那么根据各点之间的距离，我们取最近的 5 个点，分别是点 2、点 3、点 4、点 7 和点 5。很明显可以看出，因为要取固定个数的邻居，这种方法对于孤立点的计算效果不太好。当它附近没有足够多比较相似的点，就会被迫取一些不太相似的点作为邻居，这样就影响了近邻相似的程度，比如在图 5-19 中，点 1 和点 5 其实并不是很相似。

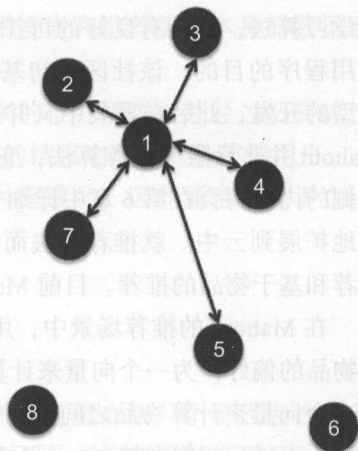


图 5-19 近邻选择方法一，通过最近的 K 个用户来确定

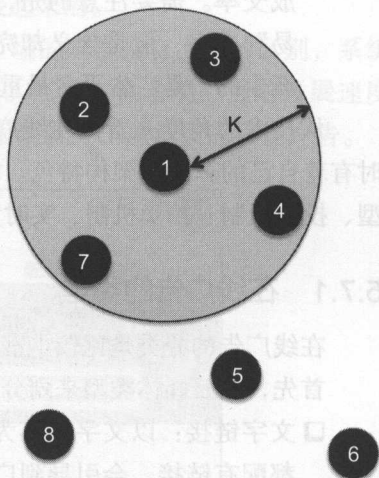
- ❑ 基于相似度阈值的近邻：与计算固定数量的近邻的原则不同，基于相似度阈值的邻居计算是对邻居的远近进行最大值的限制，落在以当前点为中心，距离为 K 的区域中的所有点都作为当前点的邻居，这种计算方法得到的



邻居个数不确定,但相似度不会出现较大的误差。如图 5-20 所示,从点 1 出发,计算相似度在  $K$  内的邻居,得到点 2、点 3、点 4 和点 7,这种方法计算出的邻居的相似度程度比前一种更合理,尤其是对孤立点的处理。这种方式要表现的就是“宁缺勿滥”,在数据稀疏的情况下效果是非常明显的。

“这样看来, Mahout 的使用还是比较直观的呢!”

“没错,你看了官网和相关书籍的介绍后,会发现 Mahout 的上手还是相对比较简单,自己搭建一个基础的推荐引擎不会有太大的难度。”小明再次给大宝打气。



## 5.7 在线广告

本章要介绍的最后一个信息检索应用是在线广告,图 5-20 近邻选择方法二,通过距离的阈值来确定它和搜索、推荐有密不可分的技术关联,但同时又存在几个很有趣的地方,例如竞价拍卖,所以还是值得我们在这里探讨一下的。

现在的广告形式日新月异,渗透到了生活的几乎每一个角落。我们在电视上、电影放映前、地铁里、商场里、车站牌上、洗手间里都会看到广告。而如今,互联网赋予了广告新的形势和定义。你在浏览美食大咖的博客时,可能就会看到牛排馆周末的优惠券。简短回顾下,20 世纪末,大型互联网公司兴起,雅虎 (Yahoo) 和美国在线 (AOL) 有着大量的用户群体和流量,他们尝试在网页中插入横幅广告 (Banner),将流量转换为收入。虽然这些广告和传统广告本质上是类似的,并没有复杂的拍卖和投放逻辑,但可以被看作是互联网广告的鼻祖。之后,Google 极大地促进了搜索和基于情景的定向广告的逐步成型。搜索广告根据用户的搜索词进行定向,然后投放相关的广告;它会基于情景的定向分析网页的内容,以及用户的历史行为,试图提供更为精准的广告。另一方面,随着广告投放精准化方向的发展,广告的拍卖和付费的模式也得到了日新月异的发展。Overture 是最早提出竞价广告的公司之一,之后 Google 公司 Adwords 的产品将这一模式推上了新的高度。随着近几年在线广告的飞速发展,为了更有效地将网络流量变现,并节省广告投放者的成本,广告交易平台应运而生。这个概念类似股票交易,这样既能帮助中小型网站解决流量变现难的问题,同时也给了广告投放者更多的自由选择。发展至今,互联网的在线广告已经有了上千亿的规模,产品形式也很多样。作为一个生态系统,目前主要包含如下角色。

- 广告投放者: 又称为广告商或广告主,即希望投放广告的商业。
- 广告: 希望广而告之的内容本身。
- 用户: 也可称为消费者、顾客。广告商希望接受广告的人群。
- 广告平台: 也可称之为广告系统,这是技术的核心。其最主要的职责就是帮助广告

投放者，找到合适的顾客，推送广告的内容，在合理控制成本的前提下提升最终的成交率。需要注意的是，我们还会提及广告交易平台，虽然只比广告平台多了“交易”二字，但是含义却完全不同。交易平台是为优化网络流量和广告投放的匹配而诞生的，是广告平台最重要的一个组成部分，但不能代表其全部。

从技术的角度来看，在线广告的很多技术都是基于搜索和推荐等检索技术的衍生，同时有着自己的算法和架构特色。本节将介绍一些基本的概念和核心要点，包括在线广告的类型、投放机制、拍卖机制、实时竞价和系统架构。

### 5.7.1 在线广告的类型

在线广告的分类是比较丰富的，为了便于理解，下面将按照不同的维度来切分。

首先，按照媒体类型来划分，可分为如下三类。

❑ 文字链接：以文字内容为主的广告。当然，和传统的线下广告不同，这些文字一般都配有链接，会引导到广告投放者的网站或活动页，这也充分体现了互联网的便捷性。由于文字链广告占用篇幅很少，而且可以跟搜索或情景模式进行配合，所以应用场景比较多，是目前的在线广告中的主流形式。

❑ 视频：视频广告和线下的视频广告非常相似，例如片头广告、片尾广告、暂停阶段的广告、正常播放中间的插片广告等。

❑ 富媒体（Rich Media）：这是内容展现力更强的一种形式，一般是采取侵入式的方式，向浏览者推送品牌性很强、创意很丰富的广告。这种广告有时候甚至可以让用户直接进行交互。

接着，按照传播渠道来划分，可分为如下三类。

❑ 网页：利用互联网最基本的要素网页，来进行广告内容的传播。这是最常见和最基本的形式。

❑ 直接营销广告 EDM（Email Direct Marketing）：通过电子邮件的方式进行宣传，图 5-21 就是寄给笔者的一封 EDM，告诉笔者有哪些热门景点的促销活动。这种传播渠道需要通过数据的累积，在一定程度上了解顾客的兴趣所在，否则非常容易沦为垃圾邮件，进入顾客或邮件服务商的黑名单。要做到这点绝非易事，试想一下过去一个月，你直接无视了多少封邮箱里的推广信件？

❑ 社交关系：如果在朋友圈里，你的好友极力推荐了一款商品，你是不是很有可能想要尝试一下？实践证明，在社交场景下推出的广告，取得了非常不错的效果。随着社交产品的飞速发展，社交广告的形式发生了巨大的变化。商家和粉丝可以用社交平台进行互动，更有甚者利用其进行定制化营销，为自己的粉丝定制商品，这种用户黏性是之前的渠道所无法比拟的。完全可以预见，社交广告未来的发展空间是无比巨大的，一定还会涌现出更多更有特色的推广方式。

下面按照平台来划分，可分为如下两类。

- PC：在移动设备普及前，自然广告的主战场就是个人计算机。目前在线广告的成熟技术和架构，也基本上是在 PC 流行的数十年里发展起来的。
- 移动端：从本质上而言，移动设备上面的广告跟 PC 端的广告没有太大的区别，系统后端的算法也基本一致，只是载体换成了移动设备。当然，移动端广告的发展速度很快，并且在产品形式上有很大的创新空间，相信很快就会出现新颖的移动广告。



图 5-21 广告营销 EDM 的样例

下面按照投放方式来划分，可分为如下两类。

- 横幅 (Banner)：起源于最早的互联网广告形式，基本上继承了传统的线下模式，一直到目前都活跃在各种网站上。这种广告一般具有固定的尺寸，内容用图片或 Flash 等方式来展现，还有一些是动态的创意素材。
- 精准定向：目前主要分为搜索广告和情景广告。搜索广告是指，在搜索过程中，搜索引擎推送给用户的互联网广告。例如，每当我们打开百度搜索引擎，输入查询，在得到搜索结果的同时，在页面的右侧和搜索结果的上方，也会产生一些推广链接。这些推广链接，就是常说的搜索广告。例如图 5-22 中给出的例子，从中我们可以看到用线框标出的部分，就是百度根据用户的输入“菲力牛排”而推送的商家广告，百度称之为“推广链接”。

另一方面，在用户没有主动输入搜索词的场景里，为了提升广告的转化率，大多数采用的是精准定向广告。这里的定向是指根据投放页面的内容，以及用户的历史行为数据，推送相关的广告。





图 5-22 精准定向广告的示例

下面按照计费模式来划分, 可分为如下五类。

- CPM (Cost per Mille): 拉丁文中 Mille 的意思是“1000 次”, CPM 即指按照千次展现计费, 当然, 实际运用中不一定要局限于千次。这种方式的本质就是按照展现计费, 系统只须让广告商的广告获得足够的曝光。至于展现之后有怎样的投资回报率, 系统是无法向广告投放者做出保证的。例如: 曝光之后有多少用户来点击? 点击之后又有多少人看到了包含广告内容的页面? 即使跳到了广告页面, 用户是否浏览并产生了最终的交易? 这些问题, 系统在 CPM 模式下是不会考虑的。因此, CPM 非常适合品牌的推广, 因为品牌一般不会只看重短期内的成交, 而是更关注长期的市场培养和品牌形象的树立。
- CPT (Cost per Time): 这是按照单位时间计费的方式。大家都知道, 央视黄金时段的广告费都是天价, 原因就是特定时间内, 观众数量非常可观, 眼球经济的效应十分明显。CPT 模式也是采用类似的原理, 针对大广告商的推广活动, 以独占时间段的方式进行推广和计费的。跟 CPM 类似, 就是广告系统不负责投放效果进行预估和管理, 只是遵守时间的限制而已。显而易见, 这种方式也很适合于中大型广告商用于增加品牌的曝光度。
- CPC (Cost per Click): 按照点击次数来计费。CPC 模式最早产生于搜索广告, 是目前应用最广泛的计费方式。使用该方式, 一方面广告系统要负责对点击率进行预估,

给出尽可能准确的预估分数；另一方面广告投放者可以参与竞价，在一定程度上控制自己的推广流量。可以看到，和 CPM 及 CPT 不同，CPC 使得广告系统和广告主的能动性都得到了一定的发挥。这种模式从投资回报率的角度上来说更为合理，也促进了广告投放和售卖方式的不断演进，后面会有更为详细的介绍。

□ CPS (Cost per Sale): CPS 方式和 CPC 非常相像，唯一不同的是以最终成交情况而不是点击率来收费。在 CPC 下，广告系统开始关注点击率这样的效果，但是仍然没有考量广告的实际成交情况，广告商自己仍然要承担不小的投放失败的风险。而 CPS 的核心就是按照成交来统计，更多的是为广告投放者考虑。

□ 其他：随着移动和社交互联网的蓬勃发展，出现了一些新的计费方式，例如按照商品和服务的收藏量、按照软件的安装数量、按照软件激活数量来付费等。

简单地总结一下，就会发现这林林总总的计费模式，其实都是为了平衡广告平台和广告投放者的利益。从 CPM、CPT、CPC 到 CPS，广告平台的收益风险越来越大，而广告商的投资风险却是越来越小。所以，实际中使用什么样的计费方式，需要从各方的利益出发综合考虑。CPC 的方式能同时照顾到流量提供方和广告主的控制权，所以目前被广泛接受，是一个不错的默认选择。

最后，按照售卖方式来划分，可分为如下两类。

□ 直接关键词竞价排名：在主流的搜索广告中，竞价机制可以抽象为一种广告位的拍卖机制。在搜索广告平台上，广告投放者会为自己的广告选择一些可能相关的关键词，并为它们出具不同的竞标价格。当广告商及需要投放的广告内容达到一定的规模后，对于某个关键词，就可能会有成千上万的广告在竞争，每条广告都会提供不同的竞价，这样就形成了相互竞拍的关系。若用户浏览时命中了相关的关键词，广告系统就会根据所有竞拍的价格，决定展示哪些广告。例如，百度的搜索广告中，展现形式是自上而下的排列条“推广链接”，在这种展现形式中，排第一位的广告提出的竞价更高，相应收取的广告费用也更高，当然点击率和回报也会相应提高。

□ 广告交易平台：目前网络流量资源有日趋分散的趋势，如果无法建立起有效的沟通机制，会使得展示广告流量购买越来越困难。大量广告商的广告无法找到与其最相匹配的流量进行投放，而大量媒体流量也无法找到最能实现其真实价值的广告，流量资源得不到合理和有效的配置。因此，近几年开始流行实时广告竞价，这种新的广告合作模式，其核心是建立一种流量交换的协议，使得媒体和广告联盟可以向全网范围的广告投放者提供其尚未售出的流量，为自己带来更高收益的同时，也提升了广告商的投放效果与回报。实时广告竞价解决了需求和供给方之间的矛盾，在广告商和媒体之间架起了一座桥梁，使得广告系统中各个群体的整合更加有效，资源配置更加优化。

纵览广告的类型划分可以看到，对于在线广告而言，非常关键的两大机制是广告的投放和拍卖，这也是广告系统区别于搜索引擎和推荐引擎的特色模块。接下来的两小节将分别

来探讨一下这两大机制。

## 5.7.2 广告投放机制

为了提升广告的效果，增加最终的转化率和成交率，时下广告的投放都力图做到精准。提到精准化，搜索和推荐技术是必不可少的。相应的，广告的投放机制也主要演变为基于搜索的广告和基于情景的广告。

### 1. 基于搜索的广告

如前所述，每当打开百度进行搜索时，在搜索结果的上方和右侧，都可能出现一些推广链接，这就是我们常说的搜索广告。你可能会很好奇，从搜索引擎收到查询，到最后展出广告，都发生了哪些事情呢？也许你已经发现，这个过程和普通的搜索非常相似。没错，我们可以认为广告系统就是另一个搜索引擎，同样需要离线索引和在线查询两个主要步骤。

因为广告相对于普通文档而言数据量要更小一些，因此在建立倒排索引的过程中，所处理的数据量往往并不大。不过这并不意味着广告的索引就一定更容易实现。广告的描述主要是来自广告商选择的竞价词，这样一来会导致索引向某些热门词汇倾斜，而另一些可能相关的词汇却被遗忘了。并且，此种情形会随着广告投放者和广告内容的增加继续恶化。因此我们需要考虑为广告抽取更多的特征，比如在哪些地区投放，属于什么产品分类，竞价词有哪些关联扩展等。

在查询过程中，除了要考虑基本的相关性原则，广告系统还需要综合考虑广告内容的质量和广告商所出的价钱，这样既可以保证消费者的用户体验，同时也能为商家带来收益。这里最大的挑战在于：如何定义广告的质量呢？广告系统的解决方式和搜索引擎里的多因素排序非常相似，通过多种因素（或特征）的提取，基于启发式或机器学习的方式，对于最终的点击率、成交率等进行预估，最终通过预估值来对广告进行排序。常见的因素包含广告商维度（投诉率、点击率、成交数等）和广告维度（标题准确性、文描的详细程度等）。

### 2. 基于情景的广告

基于搜索的广告在相关性上有不错的体验，用户较容易接受，因此应用领域是非常广泛的。不过它也存在局限性：如果不存在和用户输入高度匹配的广告，那么就无法将这些流量变现。因此，目前的广告系统还会利用推荐的技术，考虑用户浏览时的情景，以及用户本身的个性化信息，作为投放广告的依据。

这里的情景（Scenario）和之前推荐系统里提及的情景很类似，简单点说就是指用户所处的信息环境，很多个性化系统都会提及这个概念。例如，用户在浏览一篇介绍世界各地美食的博客时，当前页面就构成了一个情景；用户在购物结算页面结算时，购物车里的商品也可以构成一个情景。广告系统对于情景的利用，和推荐系统也是非常类似的，最常见的就是网页关键字提取和网页聚类。关键词的提取是根据网页的链入链接、元数据标签、文本内容等进行一定的分析，从中抽取可以匹配的关键词，然后再到广告索引中进行检索。网页聚类



利用文本聚类的技术(后面的数据挖掘章节会具体介绍),汇总了同一类型的网页内容,增加了抽取关键词的数量,提升了匹配命中的概率。

对于用户的个性化信息而言,和其他用户画像系统相似,广告系统更多的是考虑用户的背景信息和历史行为。用户背景资料是指在用户使用系统时,系统让用户提供的各种自身信息,例如性别、年龄、学历、职业、所在地区等。可是,用户背景资料往往不全,或者不够精细,推荐的效果不佳。这时就需要考虑用户的行为特征,例如搜索的关键词、特定类目上的点击、详情页的浏览行为、评价的转发和点赞等。这类行为蕴含的用户意图比较明确,也具有较高的商业价值。结合用户的背景和行为数据,广告系统就可以为用户贴上更为精准的标签,在没有精确搜索的情况下,也能根据标签进行广告的推送。

当然,推荐中的其他高级技术,例如协同过滤也是完全可以用于广告系统的。此外,基于搜索和推荐技术的广告是可以相辅相成,结合使用的。这些都给在线广告的发展开辟了新的天地。

“这样看来,广告系统和搜索引擎、推荐系统还真是差不多呢!”

“没错,很多技术都是类似的,触类旁通并不困难。不过广告系统也有自己独特的地方哦。”

### 5.7.3 广告的拍卖机制

拍卖和计费是普通搜索引擎、推荐引擎所不具备的功能,也是在线广告与它们最大的区别之一。互联网广告诞生之初,并没有高自动化的拍卖和计费系统。广告商通过线下签订商业合同的方式,从互联网媒体那里购买流量。这种参照线下交易的方式,操作起来整个流程相当复杂,广告投放者通常需要等待很长的时间才能看见广告生效。20世纪90年代,Overture公司提出了一套全新的广告竞价排名机制。广告商选择跟自己广告宣传内容可能相关的关键词,并出价对其进行竞拍。用户输入关键词时,搜索引擎按照竞价由高到低的顺序,依次展示所有匹配的广告。当用户点击了某个广告,系统就会扣除广告商竞买这个关键词所出的价格。

Overture提出的策略被称为广义一阶价格拍卖(Generalized First Price, GFP),这里的一阶价格是指广告主本身的出价。可是GFP存在明显的漏洞,随着自动竞价机器人的出现,广告商可以几乎毫无成本地尝试降低竞价,最终以极其低廉的价格竞标成功。举个例子,a、b和c三个广告商竞拍3个广告位。一开始,他们分别出价8元、6.5元和2.7元,看上去一切都很正常。突然,b广告商开始使用竞价机器人不断尝试降低价格,最终只要给出2.71元就可以保住第二名的广告位。同理,如果a广告商也使用竞价机器人,只要给出2.72元就能保住第一名的广告位。最终,广告系统的收益就会大幅缩水。GFP的最大问题就是,竞拍的价格是广告商自己决定的。Google改进了Overture的广义一阶价格,让这个价格由竞争对手来决定。听上去不可思议是吧?其实实现并不复杂。例如,对于第2位的广告,如果发生点击,那么扣费将不再按照第2位广告商的竞拍价,而是按照第3位的广告竞价、再加上一个货币最小值。第3位是没有义务也没有动力去替第2位节省开支的。在上一段的例子

中，广告商 a 和 b 发生点击的收费就分别变为了 6.51 元和 2.71 元。这是很神奇的地方，看似广告收费是向低价者靠拢，但是广告系统的实际收益反而会增加。这种我们称其为广义二阶价格（Generalized Second Price, GSP）拍卖。

由于拍卖机制的不断升级，竞价不再是一件容易的事情。因此，成熟的广告平台还会提供一些竞价工具来协助广告投放者。其本质是将一部分的信息暴露给广告商，帮助其决策。例如关键词历史的点击和成交情况、关键词和广告的相关性、根据排序模型得出的广告主的广告可能出现的位置、预估的流量和点击率，以及整体的收益等，这些在普通搜索引擎里都是无须考虑的，这点再次体现了广告系统的特色。

#### 5.7.4 广告系统架构

在纵览了在线广告的核心概念和模块后，下面来看一下一个典型的广告系统是如何组成的。图 5-23 展示了在线广告常见的系统架构。

首先看离线部分，组成部分如下。

- ❑ 广告后台：广告商通过后台系统管理投放的广告，包括增加投放的内容，拍卖出价，阅读实时和非实时报表；广告系统运营人员通过业务系统管理广告商客户。通常后台也有模块来监测投放广告的浏览量和点击量，并记录访客访问的时间、地域、来源，以及点击广告位后的行为，让广告商和运营人员可以对最终效果进行评估。
- ❑ 行为分析：广告投放系统的数据收集模块会记录用户行为数据，而行为分析模块会利用这些数据，进行特征提取、用户行为分析等操作。
- ❑ 反作弊引擎：与其他信息检索系统类似，通过点击作弊来获取非法收入的欺骗行为也存在于在线广告的平台。健壮的系统需要侦测和处罚作弊行为，包括恶意的点击、异常的访问等。离线处理中，系统需要设计人工启发式的规则，或者是基于学习的模型，为在线侦测的引擎做好准备。

然后，来看看在线部分，组成部分如下。

- ❑ 前端引擎：这部分是接收网页或移动设备发过来的广告请求，经过初步处理之后向后端传递，并在拿到后端返回的结果之后返回给请求者。
- ❑ 广告投放引擎：利用基于搜索和基于推荐的技术，对事先存放好的倒排索引进行检索，并取出对应的索引内容。同时，根据访客的习惯、喜好和需求，挖掘人群属性和行为意图，最后有针对性地投放用户感兴趣或对其有价值的广告内容。同时，系统也会存储每天产生和收集到的数据，生成广告报表，以及用来优化广告投放的算法，形成一个反馈的闭环。
- ❑ 实时转化率预估：这种预估将极大地影响广告的最终展现，对实时性要求很高，因此一般放入在线部分处理。实时预估的结果，将和离线计算的部分结合决定最终的广告位排序。
- ❑ 反作弊引擎：利用离线计算的结果，进行实时性或周期性的侦测，避免无效和非法

的计费。

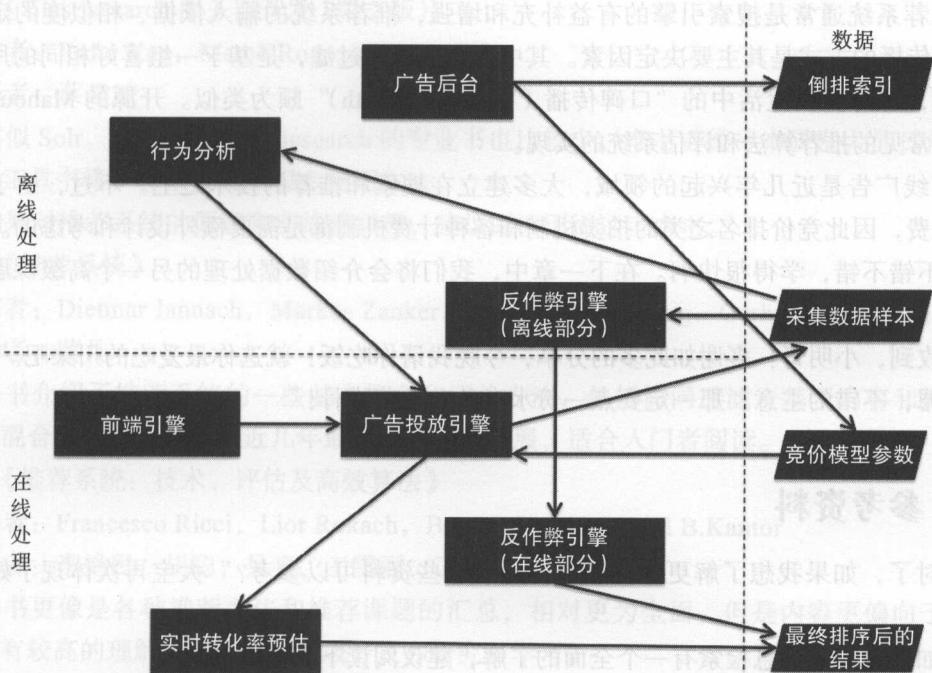


图 5-23 在线广告常见的系统架构

目前还没有一个成熟的、主流的开源项目是专门支持在线广告系统的。好在广告系统的大部分都是与搜索、推荐技术紧密相关的。因此，无论是拍卖阶段，还是展示阶段，都可以通过 Lucene、Solr、Elasticsearch 这种开源搜索系统来作为基石。而 Mahout 开源项目可以帮助离线部分的推荐算法设计和实现。

## 5.8 本章心得

通过这一章的介绍，大宝对于检索系统相关的知识有了较为全面的认识。他很激动，小明便问道：“说说看，你最大的心得体会是啥？”

“小明哥，本章介绍了大数据处理的一个高级课题——信息检索。相关性和及时性是信息检索的两大关键要素。布尔模型、向量空间模型、语言模型是常见的用于判断相关性的方法。倒排索引极大地提高了检索的效率。

搜索引擎是信息检索最基本、也是最重要的应用级系统。除了基础的相关性模型之外，搜索引擎常常还需要考虑应用领域的特征，例如 Web 网络的链接和电子商务系统的各项商品指标。开源项目 Lucene 使得开发者迅速构建一个搜索引擎成为可能。Solr 和 Elasticsearch 在 Lucene 的基础上极大地丰富和增强了搜索功能，还加入了可扩展和可伸缩的架构，适合



企业级的应用开发。

推荐系统通常是搜索引擎的有益补充和增强，推荐系统的输入依据、相似度的定义和相似度传播的方式是其主要决定因素。其中典型的协同过滤，是基于一组喜好相同的用户进行推荐，这与现实生活中的“口碑传播（Word-of-Mouth）”颇为类似。开源的 Mahout 项目提供了常见的推荐算法和评估系统的实现。

在线广告是近几年兴起的领域，大多建立在搜索和推荐的技术之上。不过，由于涉及广告付费，因此竞价排名之类的拍卖机制和各种计费机制都是需要额外设计和考虑的。”

“不错不错，学得很快啊。在下一章中，我们将会介绍数据处理的另一个高级课题：数据挖掘。”

“收到，小明哥！多谢如此多的分享，今晚我请你吃饭！就选你最爱吃的川菜吧。”

“嗯，不错的主意。那一定要点一份水煮鱼和粉蒸排骨！”

## 5.9 参考资料

“对了，如果我想了解更多的细节，还有哪些资料可以参考？”大宝再次体现了好学的精神。

“如果希望对信息检索有一个全面的了解，建议阅读下面两本书。

### □《信息检索导论》

作者：Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze

译者：王斌

### □《现代信息检索（原书第2版）》

作者：Ricardo Baeza-Yates, Berthier Ribeiro-Net

译者：黄萱菁，张奇，邱锡鹏

上述两本书都是相当经典的信息检索书籍，从入门到进阶，内容非常全面，对于理解信息检索的概念和方法都是很好的选择。其中还涉及了一些数据挖掘的篇章，认为其也是信息检索的相关部分。我们会在下一章单独阐述数据挖掘的内容。

如果对搜索引擎的原理和实战感兴趣，建议阅读下面这三本书。

### □《Lucene 实战（第2版）》

作者：Michael McCandless, Erik Hatcher, Otis Gospodnetic

译者：牛长流，肖宇

目前 Lucene 的技术已经比较完善，这本书的内容自然也是比较全面的，从搜索引擎的概念，到 Lucene 的实现和功能，都有详细的阐述。本书对于全面了解 Lucene 必不可少。

### □《Solr in Action》

作者：Trey Grainger, Timothy Potter

目前关于 Solr 的专业书不算多，这本是非常不错的指南，从入门基础知识到一些高级

功能都有介绍,也很容易上手实战。

#### □《Elasticsearch 服务器开发(第2版)》

作者: Rafal Kuc, Marek Rogozihski

译者: 蔡建斌

类似 Solr,目前关于 Elasticsearch 的专业书也比较少。本书更像一本研发人员日常开发必备的工具书籍,可以查阅到很多使用方法。

如果对推荐系统的原理和实战感兴趣,建议阅读下面这三本书。

#### □《推荐系统》

作者: Dietmar Jannach, Markus Zanker, Alexander Felfering, Gerhard Friedrich

译者: 蒋凡

本书介绍了推荐系统的一些基本概念和评价方法,包括协同过滤、基于内容和知识的推荐、混合推荐等,还有最近几年最新的案例和进展,适合入门者阅读。

#### □《推荐系统:技术、评估及高效算法》

编者: Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B.Kantor

译者: 李艳民, 胡聪, 吴宾, 王雪丽, 丁彬钊

本书更像是各种推荐方法和推荐课题的汇总,相对更为全面,但是内容更偏向于学术研究,有较高的理解难度,适合高级开发和研究员学习。

#### □《Mahout 实战》

作者: Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman

译者: 王斌, 韩冀中, 万吉

本书的内容就像标题所说的,重在“实战”,虽然对于推荐系统的概况和算法的介绍不算很系统,对于搭建推荐系统感兴趣的读者而言,还是可以作为入门的参考手册的。本书同时还介绍了 Mahout 对于聚类和分类算法的支持。

目前专门介绍在线广告的图书比较少,如下三本可供参考。

#### □《互联网广告算法和系统实践》

作者: 王勇睿

本书概要性地探讨了搜索广告算法、非搜索广告算法、实时竞价广告算法,还有对广告系统的一点点概括介绍,适合入门级的读者。

#### □《数据挖掘的应用与实践:大数据时代的案例分析》

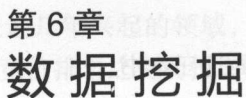
作者: 李涛等

这本书中有两个章节分别探讨了推荐系统和在线广告,可以用于入门了解。

#### □《计算广告:互联网商业变现的市场与技术》

作者: 刘鹏, 王超

作者是科研出身,全书专业性较强。如果你想对在线广告有较全面和深入的理解,这本书是不错的选择。



## 第6章

# 数据挖掘

历史总是惊人的相似。100 多年以后，由美国西海岸引领的新一轮“掘金”运动开始了，只是场所从广袤的大地搬到了无限的电子化信息库，工具从铲子和挖掘机变成了信息技术。这就是数据挖掘（Data Mining），一个新兴并且快速发展的科研领域。有时行业中也称其为数据知识发现（Knowledge Discovery in Databases），一般是指通过算法从大量的数据中挖掘有价值的模式和知识的过程。需要注意的是，数据挖掘通常与计算机的很多领域都有交集，例如统计分析、信息检索、机器学习、专家系统和模式识别等。数据挖掘是信息技术发



展的结果和趋势。自从 20 世纪 60 年代以来,数据库和信息技术从原始的文件处理逐步演变成较为复杂的数据库系统。70 年代开始,相关系统的研发逐步发展到关系型数据库和数据建模。联机事务处理(OLTP)为关系型技术的发展做出了重大的贡献。进入 80 年代中后期以来,数据库技术就逐步转向支持高级数据分析的数据仓库和数据挖掘,大量的数据库和信息存储库用于事务管理、信息检索和数据分析。随着 90 年代的到来,更多的数据不再仅仅是积累于数据库和数据仓库中,万维网和各种互联的数据库形成了基于互联网的全球信息库,我们的数据也在日益庞大。在这样的大数据时代,数据挖掘显得更为重要了。可是,数据规模的“大”不见得就必然能为我们带来“大”的价值,如何有效地从海量数据中获取价值是个难题。一方面,大数据为我们提供了更多的机遇;另一方面,大数据也向我们发出了挑战。

本章首先介绍数据挖掘的一些基本概念,阐述数据的表示和预处理的过程,然后介绍最为重要的机器学习算法和相应的系统框架,最后探讨几个常用的相关工具。

## 6.1 基本概念

为了更好地理解数据挖掘的概念,先来了解几个基本要素:数据的类型、挖掘的主题、使用的技术,以及应用的场景。

### 1. 数据的类型

对于数据的类型,大体上可以分为传统的关系型数据库和目前正在兴起的非结构化数据。

关系型数据库仍然是数据挖掘的主战场之一。研究人员可以通过它来探索趋势或研究数据模式,例如分析顾客的收入、年龄、职业,预测新顾客的信用风险等。还可以发现商品销量和价格大幅变化的原因。当多个数据源集成到一起时,通过清洗、变换、集成、装入等步骤对数据进行处理,最后就形成了数据仓库。因为数据仓库提供了多维度数据视图和汇总数据的预处理,所以非常适合联机分析处理(OLAP)。尽管数据仓库工具对于数据分析很有帮助,但是进入深入分析仍然需要更多的数据挖掘技术。多维数据挖掘允许在各种粒度上进行多维组合查询,借此发现代表知识的有趣模式。此外,数据库里的事务也是挖掘的重要对象,例如顾客的一次购物,一次旅行的订票,一次银行的转账等。考虑到不同的领域和行业,时间相关的序列数据(例如股票交易)和空间数据(例如地图)也是非常值得深入分析的。

随着互联网时代的到来,越来越多的复杂数据涌现出来,例如超文本、万维网的图状结构、社交网络、视频和音频等多媒体数据。因为受限于自身结构或数据规模等,它们很难在传统的关系型数据库里进行高效地保存或访问。正如前面介绍过的,Hadoop 及其生态系统为这种非结构化的海量数据存储和处理提供了可能。这些类型的数据通常包含更为丰富的信息,为数据挖掘提供了肥沃的土壤,也提出了更具挑战性的研究课题。

### 2. 挖掘的主题

根据目标的不同,挖掘的任务和技术也有所不同,主流的课题有发现频繁项、分类、

聚类和异常侦测，等等。

- 频繁模式和关联性：频繁模式是在数据中频繁出现的模式，包括多种类型，比如频繁项集、频繁子序列和频繁结构。频繁项集往往是指在事务数据中频繁地同时出现的商品集合，如超市中很多顾客频繁地一起购买奶瓶和尿布。频繁出现的子序列，则考虑了出现的先后顺序，例如先购买个人计算机，然后是计算机配件，最后是维修工具。而频繁结构则将频繁项目推广到树状结构和图状结构，用于考察是否存在一些子结构频繁地出现。所有这些都是为了发现数据中存在的关联性。
- 用于预测的分类和回归：分类（Classification）旨在找出描述和区分数据类的模型，以便能够使用模型预测分类信息未知的数据对象，并告诉人们它应该属于哪个类。而模型的生成，则是基于训练数据集的分析，一般分为启发式规则、决策树、数学公式和神经网络。举个例子，我们给计算机系统大量的水果，然后告诉它哪些是苹果，哪些是甜橙，通过这些样本和我们设定的建模方法，计算机学习并建立模型，最终拥有判断新数据的能力。和分类预测离散的类别标签有所不同，回归（Regression）则是建立连续值的函数模型。它是最常用的数值预测方法之一，包含基于可用数据的分布趋势识别。回想上一章提到的电子商务网站的商品排序，当有诸多因素需要考虑时，回归分析是不错的选择，它可以帮你预测商品的销量。当然，回归和分类相同，都需要训练数据集。
- 非预测性的聚类：不像分类和回归，聚类（Clustering）分析数据对象本身，而无须训练样本。很多场景下，并不存在经过标记的数据，这时可以使用聚类，根据最大化类内相似性、最小化类间相似性的原则将相似的数据归集到一个组。每个组都被称之为“群组”（Cluster），也可以叫作“簇”，那么最终的结果是群组内的数据对象之间有很高的相似性，而与其他群组中数据对象的相似度却非常低。聚类的一个经典应用是客户关系管理（CRM）系统中的用户分群，因为用户数据量通常很大，很难给每个用户群体列出具体的类别名称和定义，因此无须训练样本的聚类自然就成了首选。聚类除了具有扁平的结构外，还支持层级型的聚类。
- 异常点分析：异常点（Outlier），也称为离群点，是指一些数据对象和其他数据的行为有很明显的不一致，通常和聚类技术有很多相似之处。大部分的数据挖掘都会将异常点去除，以免影响主要的结论。然而，在某些应用中，少数表现怪异的数据反而蕴含了更有价值的信息。常见的案例包括侦测违规的用户行为，包括账户偷盗、虚假交易、信用欺诈等。

考虑到频繁模式和关联性分析在传统的商业智能领域已经有了很深入的探讨，本书就不再过多赘述，而是将更多的笔墨用于分类、回归和聚类这种机器学习的算法上。

### 3. 应用的场景

数据挖掘的价值不言而喻，其应用场景自然也是非常广泛的。对于大数据相关的领域而言，挖掘主要应用于商务智能和信息检索。

商业需要更好地理解顾客、市场、供应和资源以及竞争对手，商业智能（Business Intelligence）技术则提供商务运作的历史、现状和预测等数据，从而可以发现竞争对手的优势和劣势，留住具有高价值的客户，做出正确的业务决策。显然，数据挖掘是商业智能的核心：商业智能的联机分析处理工具依赖于数据仓库的多维度挖掘；分类和回归预测技术可以极大地协助市场分析、供应和销售；此外，客户管理时，聚类承担主要任务，它会将客户按照相似性分组。

至于信息检索（Information Retrieval），第5章已经有了大篇幅的介绍，近些年来，它和数据挖掘紧密结合，为信息系统的发展起到了相当重要的推动作用。搜索引擎中的各种链接分析、排序算法、相关性模型都可以利用数据挖掘的技术来提升精准性，例如用户查询的分类就是常见的应用。推荐引擎中最重要的用户和物品相似度的计算，也都需要使用挖掘的技术来提升效果，其中最近邻的查找和聚类在原理上也是非常接近的。

## 6.2 数据的表示和预处理

### 6.2.1 数据的表示

看过上述的简介，你也许迫不及待地想一探数据挖掘的世界。请稍微等一下，我们还有一个重要的问题需要回答：如何能够让机器理解现实中的数据？一个苹果和一个甜橙，对于人类来说看一眼就能区分，可是对于机器来说如何来分辨呢？严格意义上来说，计算机它只“懂”数字。因此，我们必须要将实际对象转换为计算机能够处理的数据对象，最为常见的方式就是将对象转化为包含多维度特征的向量。首先，让我们认识一下“特征”（Feature）这个概念，特征是一个数据字段，表示数据对象的一个属性，在不同的领域中，特征和属性、维度、变量、字段基本上都是相等的，可以互换使用。在数据库中，一个交易的时间、地点、交易物品都是交易这个数据对象的特征。回想一下第5章介绍的信息检索，一篇博客文章的标题、作者、撰写时间就是文章的特征，相应的，Lucene里的字段其实就是特征在应用系统中的具体表现形式之一。再以水果作为例子，我们可以将水果的形状、外皮颜色、外皮纹理、重量、握感、口感作为其特征。表6-1展示了为苹果、甜橙和西瓜三种水果（见图6-1）抽取的特征值。

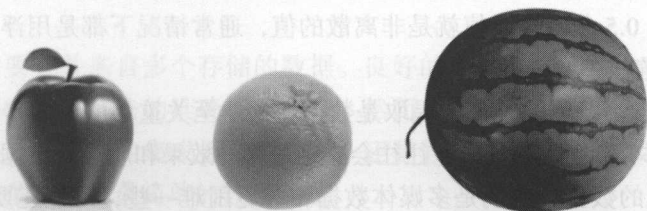


图 6-1 三种诱人的水果，如何让计算机来理解？



表 6-1    三种水果的特征值

特征 水果	形状	外皮颜色	外皮纹理	重量	握感	口感
苹果	不规则圆	红色	无	200.45 克	较硬	酸甜
甜橙	圆形	橙色	无	150.92 克	较软	甜
西瓜	椭圆形	绿色	条纹	6000.88 克	较硬	甜

当然，这样计算机还是无法理解，我们还需要将特征的值转化为数字。表 6-1 中的特征值可转化为表 6-2 中的数字。

表 6-2    将三种水果的特征值转为数字表示

特征 水果	形状 不规则圆：1 圆形：2 椭圆形：3	外皮颜色 红色：1 橙色：2 绿色：3	外皮纹理 无：1 条纹：2	重量	握感 较硬：1 较软：2	口感 酸甜：1 甜：2
苹果	1	1	1	200.45	1	1
甜橙	2	2	1	150.92	2	2
西瓜	3	3	2	6000.88	1	2

这样，三种水果对象就转化为用向量表达的三个数据对象。

苹果 = (形状 = 1, 外皮颜色 = 1, 外皮纹理 = 1, 重量 = 200.45, 握感 = 1, 口感 = 1)

甜橙 = (形状 = 2, 外皮颜色 = 2, 外皮纹理 = 1, 重量 = 150.92, 握感 = 2, 口感 = 2)

苹果 = (形状 = 3, 外皮颜色 = 3, 外皮纹理 = 2, 重量 = 6000.88, 握感 = 1, 口感 = 2)

是不是觉得这个过程似曾相识？对，在第 5 章中，我们为文章建立倒排索引时，构建了字典，随后将其概念扩展到了特征空间。其实这里的原理和概念与倒排索引是一致的。再复习一下，在实现文档检索系统的时候，用单词的 ID 作为特征 ID，并以 *tf-idf* 值或类似值作为特征的值。如此一来，一个文档集合就会转换为一组向量，每个向量代表一篇文档。这种表示忽略了单词在文章中出现的顺序，这点可以大大简化很多模型中的计算复杂度，同时保证相当高的准确性，这种处理方式通常称为词包 (Bag Of Word)。

从上述例子也可以看出，特征的值又分为离散的和连续的，离散值具有有限或无限个可数的值。在表 6-2 中，除了重量以外其他的特征都是用离散值来表示的。当然，离散值不一定要用整数来表示，例如商品的评分就可分为 1.0、1.5、2.0 一直到 4.5、5.0，这若干个档次每档都是增加 0.5 分。连续值就是非离散的值，通常情况下都是用浮点数来表示的。表 6-2 中的重量就是连续值。

“大宝，要注意，特征的定义和提取是数据挖掘中至关重要的第一步，有的时候甚至于比挖掘算法和技术本身更为关键，它往往会决定最终的效果和质量。数据库中的对象相对容易转换，非结构化的数据，特别是多媒体数据相对更困难一些，可能还要运用到图像识别、语音分析等技术来提取特征和特征值。”

“那有什么捷径可以让我快速成为这方面的专家吗?”

“其实很难有捷径,还是需要通过不断地实战来逐步积累经验。当然,后面第二部分的实战介绍,也许会给你一些启发。”

## 6.2.2 数据的预处理

在将数据转换为机器能理解的格式后,接下来就需要考虑另一个重要的问题了:如何保证数据转换后的质量呢?如果用户发现数据是有问题的,他们就会很难相信从这些数据中挖掘出来的结果。问题数据还可能导致挖掘的算法和程序陷入混乱,产生非法或不可靠的输出,对分析系统的健壮性(Robust)也提出了极大的挑战。另外,我们还需要考虑集成不同的数据源,这个处理过程可能又会引入新的噪音和冗余。因此,在该过程中还需要引入数据清洗(Data Cleaning)和数据集成(Data Integration)这两个重要的预处理步骤。

### 1. 数据清洗

现实世界中的数据往往都是不完整的,或是有杂质的。数据清洗可尝试识别异常点、填充缺失值等。现在假设一个场景,一共有1000颗水果,分为苹果、甜橙和西瓜三大类。我们请6位可爱的果农标出所有水果的特征值,其中组长1人负责200颗,其余5位组员每人负责160颗。

假设有位果农不幸是红色盲的患者,结果他测量的10个红苹果全部变成灰色外皮,这是一个明显的异常,需要将这些颜色标记有误的苹果都找出来。在识别异常点时,通常使用聚类的方法来检测。聚类会将类似的值组织成为一个群组,整个数据集可以有多个群组,例如苹果、甜橙和西瓜分为3个群组。落在所有群组之外,或者距所有群组中心都很遥远的数据对象,就可以看作是异常点(具体的聚类技术会在后面的机器学习章节详细介绍)。

再假设有位粗心的果农,忘记称其中一小部分西瓜的份量了,那该怎么办呢?最简单的方式就是丢弃这些西瓜的数据,不让它们产生价值了。有人觉得丢弃有点可惜,拿着这些瓜来逐一过称,补全重量数据。好归好,就是太费时间、太费精力了,如果想偷懒咋办?那就给一个估计值吧。怎么预估?取所有数据的平均值、中位数、最小值,最大值,可以吗?理论上固然可以,但是实际上通过苹果和甜橙来预测西瓜,好像就不太合理了。更合理的方式是,取所有称过份量的西瓜,将它们重量的平均值、中位数、最小值,或最大值等作为预估值,然后将其填充到缺乏该特征值的西瓜数据里,这就是填充缺失值的过程。

### 2. 数据集成

数据挖掘经常要合并来自多个存储的数据。良好的集成对降低冗余程度和不一致性非常有帮助,它能提升挖掘算法的准确率和效率。然而,在现实世界中,数据的语义和结构都是多样性的,这就给数据的集成提出了难题。还是假想前面的场景,我们让果农们标注1000颗水果的特征值,实现最简单的数据集成。

分工还是组长1人负责200颗,其余5位组员每人负责160颗,每个人负责1颗水果

的全部特征值标注。可惜的是，他们对于“形状”这个维度叫法不一，有位果农称其为“外形”，还有位称其为“轮廓”。好吧，我们知道这三者其实是一回事，合并的时候要做好对应，这就是对应特征的步骤。

假设有几颗甜橙放混了，被多位果农重复标注了，那么它们对应的数据对象就重复了，此时就需要去除冗余数据了。如果每颗水果都有唯一的标签，那么解决这个问题并不困难。

再假设这次为了提高效率，改为流水作业，6位果农每位只负责1维特征值的标注。最后，再将6位标注的6个维度数值结合起来。可是又出现一位粗心的果农，将分配给他的“外皮颜色”错误地理解为“外皮纹理”，这样就有两位组员重复标注了“外皮纹理”这个维度。此时，就需要去除冗余维度。根据相关系数<sup>①</sup>可以发现，这两个维度的相关性是1.0，也就是说有一个维度的数据没有提供额外的信息，可以被去除。

上面只是用最简单的例子说明了什么是对应特征、去除冗余数据和去除冗余维度，实际上还有些情况，维度并非完全重复，但是也存在信息上的重叠，导致无法提供额外的信息。比如，在一组被研究的用户中，99%以上人的国籍都是中国，那么再提供每位顾客是否说中文的信息就没有太大的价值。因此，我们还可以用相关性分析来降低维度，通过可接受的有限精度损失来换取更高的挖掘速度。相关系数的具体计算将在6.3.1节的相似度衡量中详细介绍。

“好了，大宝，到此为止你对数据挖掘的准备工作算是有了基本的了解了吧？”

“啊，小明哥，这些只是刚刚开始啊？”

“那是当然了，数据挖掘除了建立和维护数据库和数据源，进行数据清洗等预处理，还需要采用一些算法来分析其中的数据，然后才能产生价值。算法涉及的领域主要是统计学和机器学习，等等。统计学往往侧重于理论的研究，因此统计学界提供的很多技术往往都要在机器学习的领域中做进一步研究，使其变成有效的算法之后再应用到数据挖掘领域。从这个角度而言，统计学主要是通过机器学习来对数据挖掘产生推动作用的。而对于实际应用而言，机器学习则是数据分析的支撑技术，后面的章节会对机器学习的基本算法做一些介绍。”

“机器学习？这是什么？”

## 6.3 机器学习算法

好莱坞著名的系列电影《终结者》想必大家都耳熟能详了，其中主角之一“天网”让人印象深刻。之所以难忘，是因为它不是人类，而是20世纪后期人们以计算机为基础创建的人工智能防御系统，最初是用于研究军事的发展，后来自我意识觉醒，视全人类为威胁，发动了审判日。当然，这一切都是剧情里的虚构场景。那么现实生活中，机器真的可以自我学习、超越人类吗？到目前为止，还没有证据表明现实中的机器能像“天网”一样思考。但

① 相关系数在6.3.1节有所介绍。



是,机器确实能在某些课题上按照人们设定的模式进行一定程度的“学习”,这正是机器学习(Machine Learning)所关注的。机器学习是一门多领域交叉学科,涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。它专门研究计算机怎样模拟或实现人类的学习行为,以获取新的知识或技能,重新组织已有的知识结构使之不断完善,从而改善自身的性能。机器学习已经有了十分广泛的应用,例如:数据挖掘、计算机视觉、自然语言处理、生物特征识别、医学诊断,等等。

作为数据挖掘主要的分析手段,我们需要介绍机器学习中几种主流的方式和相应的算法。首先是监督式学习中的分类(Classification / Categorization)和回归(Regression)技术,然后是聚类(Clustering)这种非监督式的学习。

### 6.3.1 监督学习——分类

监督式学习(Supervised Learning),是指通过训练资料学习并建立一个模型,并依此模型推测出新的实例。训练资料是由输入数据对象和预期输出所组成的。模型的输出可以是一个离散的标签,也可以是一个连续的值,分别称为分类问题和回归分析。这样说可能还是太抽象,本小节沿用前述的水果案例,先来生动地描述一下“分类”的问题。

假想这样的场景,将1000颗水果放入一个黑箱中,我们会事先告诉果农,黑箱里只可能有苹果、甜橙和西瓜三种水果,没有其他的种类。然后每次随机摸出一颗,让果农判断它是三类中的哪一类。这就是最基本的分类问题,只提供有限的选项,而减少了潜在的复杂性和可能性。不过问题在于,计算机作为机器是不能完成人类所有的思维和决策的。分类算法试图让计算机在特定条件下,模仿人类的决策,高效率地进行分类。研究人员发现,在有限范围内做出单一<sup>①</sup>选择时,这种基于机器的方法是可行的。如果输入的是一组特征值,那么,输出的就一定是确定的选项之一。

“大宝,计算机的自动分类有很多应用场景,远远超过了划分水果,比如识别垃圾邮件、将商品挂载到产品类目、按照兴趣将顾客分组,等等,都可应用分类技术。”

“收到,看来也是一个重要的技术,应该能帮我们公司的运营节约不少人力成本。”

给出分类问题的基本概念之后,我们来理解分类的关键要素和流程。

□ 学习:指计算机通过人类标注的指导性数据,“理解”和“模仿”人类决策的过程。

□ 算法模型:分类算法通过训练数据的学习,其计算方式和最后的输出结果,称为模型。通常是指一个做决策的计算机程序及其相应的存储结构,它使得计算机的学习行为更加具体化。常见的模型有决策树、K最近邻(KNN)、朴素贝叶斯(Naive Bayes),等等。

□ 标注数据:也称为标注样本。由于分类学习是监督式的,对于每个数据对象,除了必要的特征值列表信息外,我们还必须告诉计算机它属于哪个分类。因此需要事先

<sup>①</sup> 偶尔也会让系统做出多个选择,将数据对象分到多个类中。

进行人工的标注，给每个对象指定分类的标签。在前面的水果案例中，给每个水果打上“苹果”、“甜橙”和“西瓜”的标签就是标注的过程。这点非常关键，标注数据相当于人类的老师，其质量高低直接决定机器学习的效果。值得注意的是，标注数据既可以做训练阶段的学习样本，也可以做测试阶段的预测样本。在监督式算法被大规模地应用到实际生产之前，研究人员通常会进行离线的交叉验证（Cross Validation），在这种情况下，会将大部分标注数据用在训练阶段，小部分标注数据用在测试阶段。对于交叉验证，会在第 7 章的评估方法中做进一步的阐述。在正式投入到实际的生产环境中时，往往会将所有的标注数据用于训练阶段，以提升最终的效果。

- ❑ 训练数据：也称为训练样本。带有分类标签的数据，用于学习算法的输入，以构建最终的模型。训练数据会根据是离线内测，还是在线实际生产的环境，取标注数据的子集或全集。
- ❑ 测试数据：也称为测试样本。不具备或被隐藏了标注的数据，模型会根据测试数据的特征，预测其应该具有的分类标签。在离线内测时，交叉验证会保留部分标注数据以用于测试，故意隐藏其标注值，以便于评估模型的效果。如果是在实际生产中，那么任何一个新预测的对象都是测试数据，而且只能在事后再次通过人工标注来验证其正确性。
- ❑ 训练：也可称为学习。算法模型通过训练数据进行学习的过程。
- ❑ 测试：也可称为预测。算法模型在训练完毕后，根据新数据的特征来预测其属于哪个分类的过程。

图 6-2 将如上基本要素串联起来，展示了分类学习的基本流程。

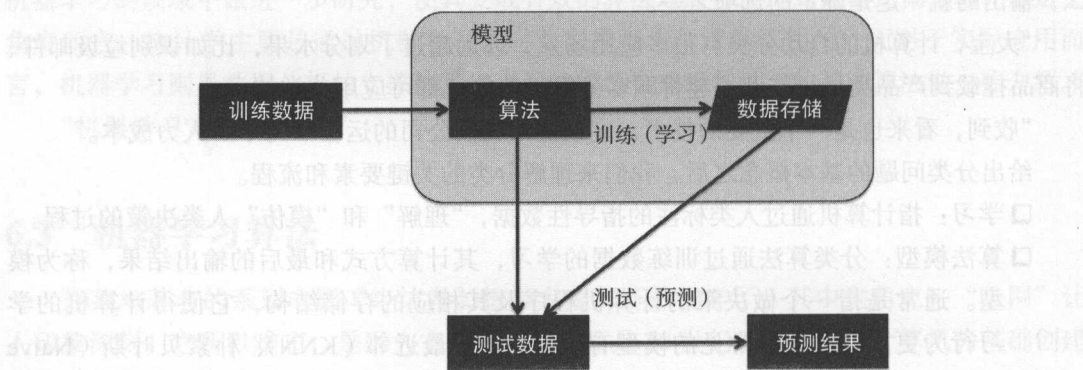


图 6-2 分类学习的基本流程

理解了这些要素和分类的过程，我们会发现除了人工标注之外，最为核心的就是分类的算法了。接下来，我们来看看几个常用的分类算法。

## 1. 决策树 (Decision Tree)

决策树学习属于归纳推理的算法之一, 应用非常广泛。它是一种逼近离散值目标函数的方法, 适用于分类问题, 其学习后的模型函数是通过一棵树来表示的, 这也是“决策树”这个名字的由来。决策树让数据实例从树的根节点走到叶子节点, 然后通过每个节点逐个地对某维度的特征值进行判断, 最后确定其分类。图 6-3 展示了水果案例中做决策的过程。

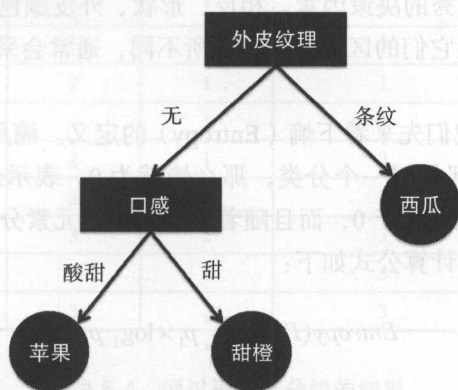


图 6-3 决策树分类的过程

当然, 通过训练数据我们可以归纳出很多种决策树, 每棵树的分类效率也是不一样的, 那如何才能发现更优的解决方案呢? 这里给出最基本的自顶向下的 ID3 贪婪搜索, 用于遍历可能的决策空间, 这也是后继优化算法 C4.5 的基础。训练的大致的流程如下:

- 1) ID3 输入 (包括训练数据、分类标签、特征列表  $TL$  等)。
- 2) 创建树的根节点。
- 3) 如果训练数据都为某一个分类, 则返回根节点树。
- 4) 如果特征列表为空了, 则返回根节点树, 分类标签取训练数据中最多的那个标签。

如果上述所有条件都不满足, 那么:

- a) 在特征列表中选择最有区分能力的一个特征  $T$ 。
- b) 根节点的决策采用特征  $T$ 。

对于特征  $T$  的每个可能的取值  $v_i$ , 执行如下步骤:

- 1) 在根节点下面增加一个新的分支, 对应  $T$  值为  $v_i$ 。
- 2) 将训练数据中  $T$  值为  $v_i$  的样本命名为  $E_{v_i}$ 。
- 3) 如果  $E_{v_i}$  为空, 那就在这个新的分支下面增加一个叶子节点, 分类标签取训练数据中最多的那个标签。
- 4) 如果  $E_{v_i}$  不为空, 那就在这个新的分支下面增加一个子树。该子树由递归<sup>①</sup>调用本算法的第 1 步来生成, 输入是“训练数据子集  $E_{v_i}$ 、分类标签、 $TL$  中去除  $T$  之后剩下的列表”。

① 递归是常用的编程技巧, 用于逐步拆解复杂的问题。感兴趣的读者可以参考程序设计类图书。



“小明哥，上面的步骤看起来不难，但是最为关键的一步‘在特征列表中选择最有区分能力的一个特征  $T$ ’，这个要如何理解和确定呢？”

“我们所说的特征‘最有区分能力’，指的是利用这维特征可以对最多的数据进行分类。对于苹果、甜橙和西瓜，如果有 1 个特征是它们是否可以食用，那么靠这个特征是无法将它们区分为两组或三组水果的，因为它们都是可以吃的。这个特征没有提供任何有价值的线索，也必然不能成为一个优秀的决策因素。相反，形状、外皮颜色、外皮纹理等都可以帮助我们区分不同类的水果，但它们的区分能力也有所不同，通常会采用信息增益（Information Gain）来衡量。”

在了解信息增益前，我们先来看下熵（Entropy）的定义。熵用于刻画给定集合的纯度，如果一个集合里的元素全部是同一个分类，那么熵就为 0，表示最纯净。如果元素分布在不同的分类里，那么熵的值就大于 0，而且随着分类越多，元素分布的越均匀，熵值也就越大，表示混乱程度越高。其计算公式如下：

$$Entropy(P) = - \sum_{i=1}^n p_i \times \log_2 p_i$$

其中  $n$  表示集合中分类的数量， $p_i$  表示属于第  $i$  个分组的元素在集合中的占比。假设 1000 个水果里，有 300 个苹果，300 个甜橙，400 个西瓜，那么这个分类的熵值约是 1.57：

$$Entropy(P) = - (0.3 \times \log_2 0.3 + 0.3 \times \log_2 0.3 + 0.4 \times \log_2 0.4) \approx 1.57$$

理想状况下，如果分类完毕后，每组都是最纯净的，熵值都为 0，那么，决策树分类问题就转化为如何快速地将熵从 1.57 下降到可能的最小值<sup>①</sup>，从而最终达到分类的效果。自然，每次选择特征时，我们都希望熵值下降的最多，这个就是信息增益，公式定义如下：

$$Gain(P, T) = Entropy(P) - \sum_{v \in Value(T)} \frac{|P_v|}{|P|} Entropy(P_v)$$

其中  $T$  表示当前选择的特征， $Entropy(P)$  表示选择特征  $T$  之前的熵， $Entropy(P_v)$  表示特征  $T$  取值为  $v$  分组的熵， $\sum_{v \in Value(T)} \frac{|P_v|}{|P|} Entropy(P_v)$  表示选择  $T$  做决策之后，各种取值加权平均后整体的熵。 $Gain(P, T)$  表示两个熵值之差，越大表示获益越多，应该选择这维特征  $T$ 。

为了更好地理解，我们来仔细分析一下水果的案例。表 6-3 对表 6-2 进行了扩展，展示了包含 10 个水果的标注数据集。需要注意的是，针对决策树的特点，我们将重量由连续值转为了离散值。

我们来看看作为决策树的根节点，第一个特征应该如何选取。为了便于公式表达，各个特征的英文名字对应如下：形状→Shape，外皮颜色→Color，外皮纹理→Texture，重量→Weight，握感→Feel，口感→Taste。如果按照形状来划分，那么结果如表 6-4 所示。

① 为什么不是 0 呢？原因是现实中很多训练的数据和特征不足以让我们训练出一个完全精确的模型。

表 6-3 10 颗水果的特征值转化为数字表示

特征 水果	形状 不规则圆: 1 圆形: 2 椭圆形: 3	外皮颜色 红色: 1 橙色: 2 绿色: 3	外皮纹理 无: 1 条纹: 2	重量 小于 200g: 1 200g 和 500g 间: 2 大于 500g: 3	握感 较硬: 1 较软: 2	口感 酸甜: 1 甜: 2
苹果 a	1	1	1	2	1	1
苹果 b	1	1	1	1	1	1
苹果 c	2	3	1	1	2	1
甜橙 a	2	2	1	1	2	2
甜橙 b	2	2	1	2	2	2
甜橙 c	1	2	1	2	1	1
西瓜 a	3	3	2	3	1	2
西瓜 b	3	3	2	3	1	1
西瓜 c	3	3	2	3	1	2
西瓜 d	1	3	2	3	2	2

表 6-4 通过形状来分组的结果

	不规则圆: Shape-1 数量: 4 占比: 40%	圆形: Shape-2 数量: 3 占比: 30%	椭圆形: Shape-3 数量: 3 占比: 30%
组员	苹果 a 苹果 b 甜橙 c 西瓜 d (组内占比: 苹果 50%, 甜橙 25%, 西瓜 25%)	苹果 c 甜橙 a 甜橙 b (组内占比: 苹果 33%, 甜橙 67%)	西瓜 a 西瓜 b 西瓜 c (组内占比: 西瓜 100%)

计算公式如下:

$$Entropy(Shape-1) = -(0.5 \times \log_2 0.5 + 0.25 \times \log_2 0.25 + 0.25 \times \log_2 0.25) = 1.5$$

$$Entropy(Shape-2) = -(0.33 \times \log_2 0.33 + 0.67 \times \log_2 0.67) \approx 0.92$$

$$Entropy(Shape-3) = -(1.0 \times \log_2 1.0) = 0$$

其中  $Entropy(Shape-1)$ 、 $Entropy(Shape-2)$ 、 $Entropy(Shape-3)$  分别表示了不规则圆、圆形、椭圆形 3 个分组的熵。那么选择形状这个特征所带来的信息增益计算为:

$$\begin{aligned}
 Gain(P, Shape) &= Entropy(P) - \sum_{v \in Value(Shape)} \frac{|P_v|}{|P|} Entropy(P_v) \\
 &= Entropy(P) - (0.4 \times Entropy(Shape-1) + 0.3 \times Entropy(Shape-2) \\
 &\quad + 0.3 \times Entropy(Shape-3)) \\
 &= 1.57 - (0.4 \times 1.5 + 0.3 \times 0.92 + 0.3 \times 0) \\
 &= 0.69
 \end{aligned} \tag{6-1}$$

以此类推，选择其他特征，分别计算信息增益，结果为：

$$\begin{aligned} Gain(P, Color) &= Entropy(P) - \sum_{v \in Value(Color)} \frac{|P_v|}{|P|} Entropy(P_v) \\ &= 1.57 - (0.2 \times 0 + 0.3 \times 0 + 0.5 \times 0.72) \\ &= 1.21 \end{aligned}$$

$$\begin{aligned} Gain(P, Texture) &= Entropy(P) - \sum_{v \in Value(Texture)} \frac{|P_v|}{|P|} Entropy(P_v) \\ &= 1.57 - (0.4 \times 1.0 + 0.6 \times 0) \\ &= 1.17 \end{aligned}$$

$$\begin{aligned} Gain(P, Weight) &= Entropy(P) - \sum_{v \in Value(Weight)} \frac{|P_v|}{|P|} Entropy(P_v) \\ &= 1.57 - (0.3 \times 0.92 + 0.3 \times 0.92 + 0.4 \times 0) \\ &= 1.02 \end{aligned}$$

$$\begin{aligned} Gain(P, Feel) &= Entropy(P) - \sum_{v \in Value(Feel)} \frac{|P_v|}{|P|} Entropy(P_v) \\ &= 1.57 - (0.6 \times 1.46 + 0.4 \times 1.5) \\ &= 0.09 \end{aligned}$$

$$\begin{aligned} Gain(P, Taste) &= Entropy(P) - \sum_{v \in Value(Taste)} \frac{|P_v|}{|P|} Entropy(P_v) \\ &= 1.57 - (0.5 \times 1.37 + 0.5 \times 1.76) \\ &= 0.003 \end{aligned} \tag{6-2}$$

从公式（6-1）和公式（6-2）可以看出在这个数据集合里，外皮颜色具有最佳的区分能力。那么决策树第一步的根节点就应该选择这个特征。选择后，我们可以看到如图 6-4 所示的决策树和分类。

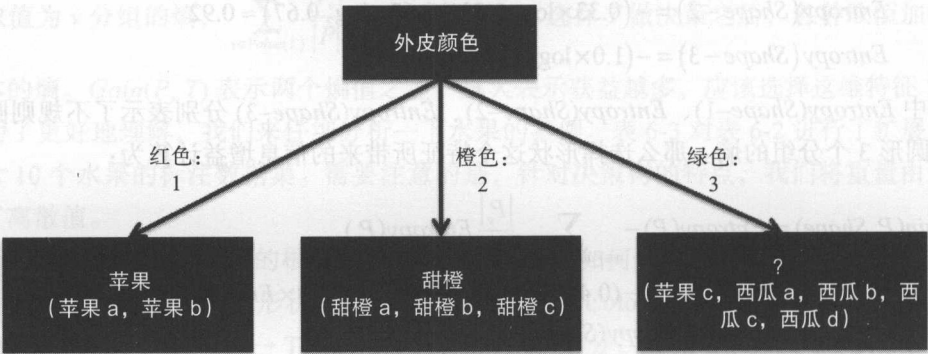


图 6-4 选择第一个特征后的决策树



从图 6-4 中可以看出,在这个标注数据集合上,红色和橙色已经可以确定苹果和甜橙的分类。但是绿色还无法完全分辨是苹果 c 还是西瓜。因此,在最右侧“?”这个节点上还需要利用递归的方式进一步在剩下的特征中选取,标准同样是根据信息增益。只是由于数据集合不再是全部 10 个水果,而只限于苹果 c,西瓜 a、b、c、d 这 5 个了,因此要重新根据公式 (6-1) 和公式 (6-2) 进行计算。直到信息增益为 0,或者是没有更多的训练数据为止。如此形成的决策树,就是分类模型的一种。对于新的数据,只要具备形状、外皮颜色等这些特征值,就可以在这棵树里游走,在每个节点处依据相应的特征来判断走向,以此找到最终的分类标签。

“哦,那这样看来归纳性质的学习计算量不小啊。”

“是的,大宝,你可以尝试一下,看看‘?’这个节点下一步会选择哪个特征作为判定条件。”

## 2. 朴素贝叶斯 (Naive Bayes)

朴素贝叶斯分类是实用性很强的一种分类方法,在某些领域内其性能与决策树学习相当。在理解朴素贝叶斯分类之前,我们先来复习一下第 5 章信息检索语言模型中所提到的贝叶斯理论。贝叶斯决策理论是主观贝叶斯派归纳理论的重要组成部分。贝叶斯决策就是在信息不完整的情况下,对部分未知的状态用主观概率来估计,然后用贝叶斯公式对发生概率进行修正,最后再利用期望值和修正概率做出最优决策。其基本思想是:

- 1) 已知类条件概率密度参数表达式和先验概率。
- 2) 利用贝叶斯公式转换成后验概率。
- 3) 根据后验概率大小进行决策分类。

最主要的贝叶斯公式如下:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

其中,在未知事件里, $B$  出现时  $A$  出现的后验概率在主观上等于已有事件中  $A$  出现时  $B$  出现的先验概率值乘以  $A$  出现的先验概率值,然后除以  $B$  出现的先验概率值所得到的最终结果。这就是贝叶斯的核心:用先验概率估计后验概率。具体到分类模型中,上述公式可以重写为:

$$P(c|f) = \frac{P(f|c) \times P(c)}{P(f)}$$

对上述公式的理解如下:将  $c$  看作一个分类,将  $f$  看作样本的特征之一,此时等号左边  $P(c|f)$  为待分类样本中出现特征  $f$  时该样本属于类别  $c$  的概率,而等号右边  $P(f|c)$  是根据训练数据统计得到分类  $c$  中出现特征  $f$  的概率, $P(c)$  是分类  $c$  在训练数据中出现的概率,最后  $P(f)$  是特征  $f$  在训练样本中出现的概率。

看完贝叶斯公式后,朴素贝叶斯就很容易理解了。朴素贝叶斯是基于一个简单假设所

建立的一种贝叶斯方法，它假定数据对象的不同特征对其归类时的影响是相互独立的。此时若数据对象  $o$  中同时出现特征  $f_i$  与  $f_j$ ，则对象  $o$  属于类别  $c$  的概率为：

$$\begin{aligned} P(c|o) &= P(c|f_i, f_j) = P(c|f_i) \times P(c|f_j) \\ &= \frac{P(f_i|c) \times P(c)}{P(f_i)} \times \frac{P(f_j|c) \times P(c)}{P(f_j)} \end{aligned}$$

下面以表 6-3 中的数据来建立朴素贝叶斯模型，其中，苹果分类共包含 3 个数据实例，对于形状而言，出现 2 次不规则圆、1 次圆形和 0 次椭圆形，因此各自的统计概率为 0.67、0.33 和 0.00。以此类推，所有的统计结果如表 6-5 所示。

表 6-5 每个分类中各个特征的统计先验概率

特征 \ 水果	形状 不规则圆：1 圆形：2 椭圆形：3	外皮纹理 红色：1 橙色：2 绿色：3	外皮颜色 无：1 条纹：2	重量 小于 200g：1 200g 和 500g 间：2 大于 500g：3	握感 较硬：1 较软：2	口感 酸甜：1 甜：2
苹果	1: 0.67 2: 0.33 3: 0.00	1: 0.67 2: 0.00 3: 0.33	1: 1.00 2: 0.00	1: 0.67 2: 0.33 3: 0.00	1: 0.67 2: 0.33	1: 1.00 2: 0.00
甜橙	1: 0.33 2: 0.67 3: 0.00	1: 0.00 2: 1.00 3: 0.00	1: 1.00 2: 0.00	1: 0.33 2: 0.67 3: 0.00	1: 0.33 2: 0.67	1: 0.33 2: 0.67
西瓜	1: 0.25 2: 0.00 3: 0.75	1: 0.00 2: 0.00 3: 1.00	1: 0.00 2: 1.00	1: 0.00 2: 0.00 3: 1.00	1: 0.75 2: 0.25	1: 0.25 2: 0.75
总共	1: 0.40 2: 0.30 3: 0.30	1: 0.20 2: 0.30 3: 0.50	1: 0.60 2: 0.40	1: 0.30 2: 0.30 3: 0.40	1: 0.60 2: 0.40	1: 0.50 2: 0.50

对于表 6-5 中出现的 0.00 概率，在做贝叶斯公式中的乘积计算时，会导致结果为 0。因此我们通常取一个比这个数据集里最小统计概率还要小的极小值，来代替零概率。这里取 0.01，表 6-5 就会替换为表 6-6 中的数据。这种技巧也会运用于补充新数据中从未出现的特征值，称为平滑 (Smoothing)。

表 6-6 将 0.00 替换为一个极小值 0.01

特征 \ 水果	形状 不规则圆：1 圆形：2 椭圆形：3	外皮颜色 红色：1 橙色：2 绿色：3	外皮纹理 无：1 条纹：2	重量 小于 200g：1 200g 和 500g 间：2 大于 500g：3	握感 较硬：1 较软：2	口感 酸甜：1 甜：2
苹果 (Apple)	1: 0.67 2: 0.33 3: 0.01	1: 0.67 2: 0.01 3: 0.33	1: 1.00 2: 0.01	1: 0.67 2: 0.33 3: 0.01	1: 0.67 2: 0.33	1: 1.00 2: 0.01

(续)

特征 水果	形状 不规则圆: 1 圆形: 2 椭圆形: 3	外皮颜色 红色: 1 橙色: 2 绿色: 3	外皮纹理 无: 1 条纹: 2	重量 小于 200g: 1 200g 和 500g 间: 2 大于 500g: 3	握感 较硬: 1 较软: 2	口感 酸甜: 1 甜: 2
甜橙 (Orange)	1: 0.33 2: 0.67 3: 0.01	1: 0.01 2: 1.00 3: 0.01	1: 1.00 2: 0.01	1: 0.33 2: 0.67 3: 0.01	1: 0.33 2: 0.67	1: 0.33 2: 0.67
西瓜 (Watermelon)	1: 0.25 2: 0.01 3: 0.75	1: 0.01 2: 0.01 3: 1.00	1: 0.01 2: 1.00	1: 0.01 2: 0.01 3: 1.00	1: 0.75 2: 0.25	1: 0.25 2: 0.75
总共	1: 0.40 2: 0.30 3: 0.30	1: 0.20 2: 0.30 3: 0.50	1: 0.60 2: 0.40	1: 0.30 2: 0.30 3: 0.40	1: 0.60 2: 0.40	1: 0.50 2: 0.50

这样, 假设我们有一个新的水果, 它的形状是圆形, 口感是甜的, 那么它属于苹果、甜橙和西瓜的概率分别是 0.198%、26.934% 和 0.798%, 具体计算过程如下:

$$\begin{aligned}
 P(\text{apple} | o) &= P(\text{apple} | \text{shape}-2, \text{taste}-2) \\
 &= P(\text{apple} | \text{shape}-2) \times P(\text{apple} | \text{taste}-2) \\
 &= \frac{P(\text{shape}-2 | \text{apple}) \times P(\text{apple})}{P(\text{shape}-2)} \times \frac{P(\text{taste}-2 | \text{apple}) \times P(\text{apple})}{P(\text{taste}-2)} \\
 &= \frac{0.33 \times 0.30}{0.30} \times \frac{0.01 \times 0.30}{0.50} \\
 &= 0.33 \times 0.006 \\
 &= 0.00198
 \end{aligned}$$

$$\begin{aligned}
 P(\text{orange} | o) &= \frac{P(\text{shape}-2 | \text{orange}) \times P(\text{orange})}{P(\text{shape}-2)} \times \frac{P(\text{taste}-2 | \text{orange}) \times P(\text{orange})}{P(\text{taste}-2)} \\
 &= \frac{0.67 \times 0.30}{0.30} \times \frac{0.67 \times 0.30}{0.50} \\
 &= 0.67 \times 0.402 \\
 &= 0.26934
 \end{aligned}$$

$$\begin{aligned}
 P(\text{watermelon} | o) &= \frac{P(\text{shape}-2 | \text{watermelon}) \times P(\text{watermelon})}{P(\text{shape}-2)} \\
 &\quad \times \frac{P(\text{taste}-2 | \text{watermelon}) \times P(\text{watermelon})}{P(\text{taste}-2)} \\
 &= \frac{0.01 \times 0.40}{0.30} \times \frac{0.75 \times 0.40}{0.50} \\
 &= 0.0133 \times 0.6 \\
 &= 0.00798
 \end{aligned}$$



考虑到概率乘积通常都非常小，在实际运用中还会用一些数学手法进行转换，但是原理仍然保持不变。

3. K 最近邻 (K-Nearest Neighbor)

归纳性质的决策树和贝叶斯理论的分类器，在训练阶段需要较大的计算量，而在测试阶段的计算量非常小。而有一种基于实例的归纳学习却恰恰相反，训练时几乎没有计算负担，但是在面对新数据对象时却有很大的计算开销。基于实例的方法最大的优势在于概念简明易懂，这里介绍最基础的 K 最近邻分类法 (KNN)。

KNN 分类算法其核心思想是，假定所有的数据对象都对应于  $n$  维空间中的某个点，如果一个数据对象在特征空间中的  $k$  个最相邻对象中的大多数属于某一个类别，则该对象也属于这个类别，并具有这个类别上样本的特性。KNN 方法在类别决策时，只与极少量的相邻样本有关。由于主要是靠周围有限的邻近样本，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合。图 6-5 表示了水果案例中的 K 近邻算法的简化图。因为水果对象的特征维度远远超过 2 维，所以这里将多维空间中的点简单地投影到二维空间，以便于图示和理解。图 6-5 中  $N$  设置为 5，待判定的新数据对象“?”最近的 5 个邻居中，有 3 个甜橙、1 个苹果和 1 个西瓜，因此取最多数的甜橙作为该未知对象的分类标签。

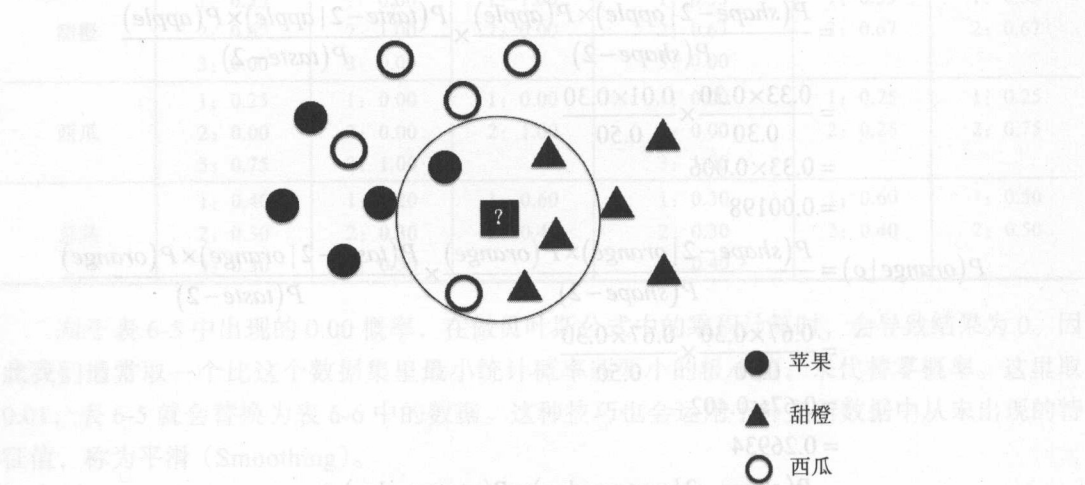


图 6-5 新的数据对象被 KNN 判定为甜橙， $N$  取值为 5

KNN 基本无须训练，这里给出预测算法的大致流程：

- 1) KNN 输入训练数据、分类标签、特征列表  $TL$ 、相似度定义、 $k$  设置等数据。
- 2) 给定等待预测的新数据。
- 3) 在训练数据集中寻找最近的  $k$  个邻居。
- 4) 统计  $k$  个邻居中最多数的分类标签，赋值给给定的新数据，公式如下：

$$\text{label}(x_{\text{new}}) \leftarrow \arg \max_{l \in L} \sum_{i=1}^k \delta(l, \text{label}(x_i))$$

其中  $x_{\text{new}}$  表示待预测的新数据对象,  $l$  表示分类标签,  $L$  表示分类标签的集合,  $x_i$  表示  $k$  个邻居中的第  $i$  个对象。如果  $x_i$  的分类标签  $\text{label}(x_i)$  和  $l$  相等, 那么  $\delta(l, \text{label}(x_i))$  取值为 1, 否则取值为 0。我们可以对 KNN 算法进行一个直观的改进, 根据每个近邻和待测点  $x_{\text{new}}$  的距离, 将更大的权值赋给更近的邻居。比如, 可以根据每个近邻于  $x_{\text{new}}$  的距离平方的倒数来确定近邻的“选举权”, 改进公式如下:

$$\text{label}(x_{\text{new}}) \leftarrow \arg \max_{l \in L} \sum_{i=1}^k w_i \times \delta(l, \text{label}(x_i))$$

$$w_i = \frac{1}{d(x_{\text{new}}, x_i)^2}$$

下面再次以表 6-3 的数据集合为例, 假设前 9 个水果是用于训练的, 最后一个西瓜 c 用于测试, 并用“?”表示西瓜 c。那么它们的特征向量如下:

$$\text{apple}_a = (1, 1, 1, 2, 1, 1)$$

$$\text{apple}_b = (1, 1, 1, 1, 1, 1)$$

$$\text{apple}_c = (2, 3, 1, 1, 2, 1)$$

$$\text{orange}_a = (2, 2, 1, 1, 2, 2)$$

$$\text{orange}_b = (2, 2, 1, 2, 2, 2)$$

$$\text{orange}_c = (1, 2, 1, 2, 1, 1)$$

$$\text{watermelon}_a = (3, 3, 2, 3, 1, 2)$$

$$\text{watermelon}_b = (3, 3, 2, 3, 1, 1)$$

$$\text{watermelon}_c = (3, 3, 2, 3, 1, 2)$$

$$? = (1, 3, 2, 3, 2, 2)$$

然后, 我们计算“?”向量和所有训练样本的相似度, 这里以欧氏距离为例。“?”和各个训练样本的欧氏距离分别为:

$$\begin{aligned} \text{ED}(\text{apple}_a, ?) &= \sqrt{(1-1)^2 + (1-3)^2 + (1-2)^2 + (2-3)^2 + (1-2)^2 + (1-2)^2} \\ &= 2.83 \end{aligned}$$

$$\begin{aligned} \text{ED}(\text{apple}_b, ?) &= \sqrt{(1-1)^2 + (1-3)^2 + (1-2)^2 + (1-3)^2 + (1-2)^2 + (1-2)^2} \\ &= 3.32 \end{aligned}$$

$$\begin{aligned} \text{ED}(\text{apple}_c, ?) &= \sqrt{(2-1)^2 + (3-3)^2 + (1-2)^2 + (1-3)^2 + (2-2)^2 + (1-2)^2} \\ &= 2.65 \end{aligned}$$

$$ED(orange_a, ?) = \sqrt{(2-1)^2 + (2-3)^2 + (1-2)^2 + (1-3)^2 + (2-2)^2 + (2-2)^2} \\ = 2.65$$

$$ED(orange_b, ?) = \sqrt{(2-1)^2 + (2-3)^2 + (1-2)^2 + (2-3)^2 + (2-2)^2 + (2-2)^2} \\ = 2$$

$$ED(orange_c, ?) = \sqrt{(1-1)^2 + (2-3)^2 + (1-2)^2 + (2-3)^2 + (1-2)^2 + (1-2)^2} \\ = 2.24$$

$$ED(\square a \square \in \square elon_a, ?) = \sqrt{(3-1)^2 + (3-3)^2 + (2-2)^2 + (3-3)^2 + (1-2)^2 + (2-2)^2} \\ = 2.24$$

$$ED(\square a \square \in \square elon_b, ?) = \sqrt{(3-1)^2 + (3-3)^2 + (2-2)^2 + (3-3)^2 + (1-2)^2 + (1-2)^2} \\ = 2.45$$

$$ED(\square a \square \in \square elon_c, ?) = \sqrt{(3-1)^2 + (3-3)^2 + (2-2)^2 + (3-3)^2 + (1-2)^2 + (2-2)^2} \\ = 2.24$$

这里距离越小，相似度越高。因此，如果 K 值取 1，那么最相似的是 *orange<sub>b</sub>*，将会错误地将西瓜 *c* 当成甜橙。如果 K 值取 5，那么最相似的是 *orange<sub>b</sub>*、*orange<sub>c</sub>*、*watermelon<sub>a</sub>*、*watermelon<sub>b</sub>* 和 *watermelon<sub>c</sub>*，最终会正确地分到西瓜这个类别中。

“小明哥，这和基于用户的协同过滤推荐算法中，计算用户的最近邻好像很类似啊。”

“没错，这两者的思路是相同的，略有不同的是，KNN 通常是说距离，协同过滤通常是说相似度，在这里认为是可替换的概念，只是距离和相似度是成反比的。因此，我们也可以模仿基于用户的协同过滤，不是通过设置 *N* 这个参数来确定最近邻，而是通过设置相似度或距离的阈值，来确定最近邻的集合。”

“那这样看来，距离或相似度的计算还很重要啊。”

“是的，从前面的介绍中，我们不难发现，如果相似度的衡量不准确，那么再精细的 KNN 算法也很难挖掘出有价值的信息。通常 KNN 算法采用欧氏距离来定义相似度，但实际上‘相似度’的概念更宽泛，其定义不一定要限于欧氏距离。我们扩展一下再介绍几种常见的衡量标准。第 5 章中，已经简单地介绍了 Mahout 几种关于相似度计算的实现，这里详细解释下其中的数学原理。常用的相似度衡量可以分为两大类：按照距离和按照相关性。”

按照距离衡量的方法包括欧氏距离、曼哈顿距离和余弦相似度，它们将两个数据对象的向量想象为 *n* 维空间中的点，然后计算它们之间的距离或夹角。



□ 欧氏距离 (Euclidean Distance) 相似度: 欧氏距离是通常采用的一种距离定义, 指在  $n$  维空间中两个点之间的真实距离。在二维和三维空间中的欧氏距离就是两点之间的实际距离。利用欧氏距离定义的相似度, 取值范围在  $[0, 1]$ , 其值越小, 说明距离越近, 相似度就越大。计算公式如下:

$$ED(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

其中  $n$  表示向量维度,  $x_i$  表示第一个向量的第  $i$  维数值,  $y_i$  表示第二个向量的第  $i$  维数值。

□ 余弦相似度: 和向量空间模型 (VSM) 类似, 它是利用多维空间两点与所设定的点形成夹角的余弦值来定义相似度的 (如图 6-6 所示)。取值范围在  $[-1, 1]$ , 当两个向量的方向重合时夹角余弦取最大值 1, 当两个向量的方向完全相反时夹角余弦取最小值 -1。值越大, 说明夹角越小, 两点相距就越近, 相似度就越大。计算公式如下:

$$\text{Cosine}(X, Y) = \frac{\sum_{i=1}^n (x_i \times y_i)}{\sqrt{\sum_{i=1}^n x_i^2 \times \sum_{i=1}^n y_i^2}}$$

其中  $n$  表示向量维度,  $x_i$  表示第一个向量的第  $i$  维数值,  $y_i$  表示第二个向量的第  $i$  维数值。

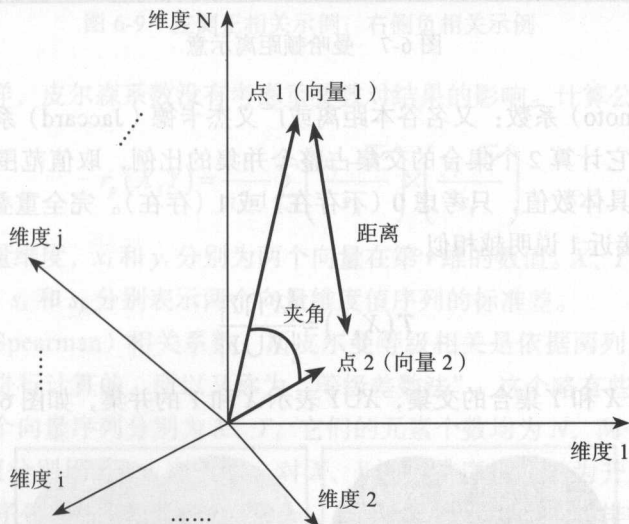


图 6-6 向量空间中的欧氏距离和夹角

□ 曼哈顿距离 (Manhattan Distance): 从名字就可以猜出这种距离的计算方法了。想象你在美国人口稠密的曼哈顿地区, 从一个十字路口开车到另外一个十字路口, 驾驶距离是两点间的直线距离吗? 显然不是, 因为你无法穿越大楼。实际驾驶距离就

是曼哈顿距离（如图 6-7 所示），而这也是其名称的来源，有时也称为城市街区距离（City Block Distance）。计算公式如下：

$$MD(X,Y)=\sum_{i=1}^n|x_i-y_i|$$

其中  $n$  表示向量维度， $x_i$  表示第一个向量的第  $i$  维数值， $y_i$  表示第二个向量的第  $i$  维数值。

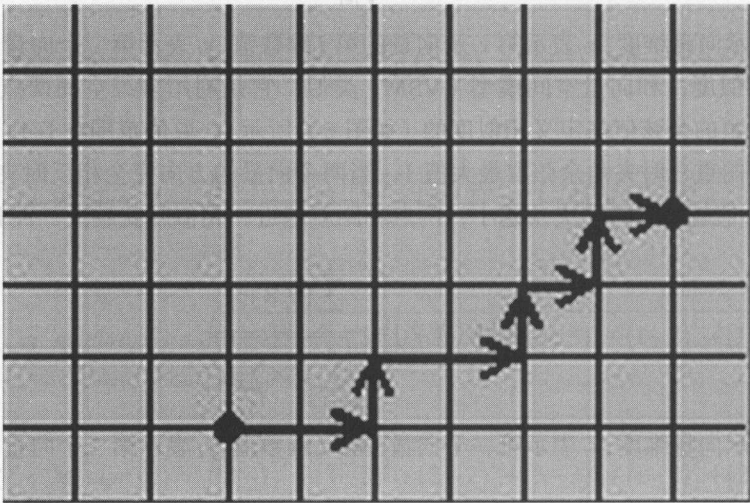


图 6-7 曼哈顿距离示意

❑ 谷本（Tanimoto）系数：又名谷本距离或广义杰卡德（Jaccard）系数，是对杰卡德系数的扩展。它计算 2 个集合的交集占整个并集的比例，取值范围在  $[0, 1]$ ，忽略了每个维度的具体数值，只考虑 0（不存在）或 1（存在）。完全重叠时为 1，无重叠项时为 0，越接近 1 说明越相似。

$$T(X,Y)=\frac{|X\cap Y|}{|X\cup Y|}$$

其中  $X\cap Y$  表示  $X$  和  $Y$  集合的交集， $X\cup Y$  表示  $X$  和  $Y$  的并集，如图 6-8 所示。

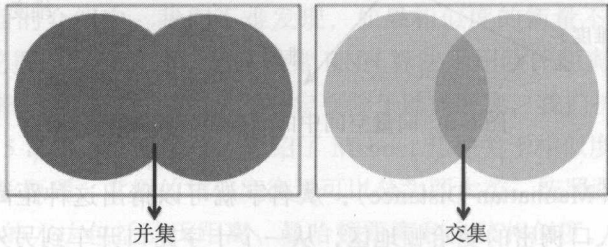


图 6-8 谷本系数中用到的集合并与交

除了按照距离的方式,还可以按照相关性来计算,将两个数据对象的向量想象为一个数值序列,然后看序列之间变化的相关性。

□ 皮尔森 (Pearson) 相关系数: 用来反映两个变量线性相关程度的统计量。取值范围在  $[-1, 1]$ , 绝对值越大, 说明相关性越高, 负数表示负相关。正相关性越高, 可以认为相似度越高, 反之, 负相关性越高, 可以认为相似度越低。图 6-9 表示了正相关和负相关的含义。左侧  $X$  曲线和  $Y$  曲线有非常近似的变化趋势, 当  $X$  上升  $Y$  往往也是上升的, 当  $X$  下降  $Y$  往往也下降, 这表示两者有较强的正相关性。右侧  $X$  和  $Y$  两者正好相反, 当  $X$  上升的时候,  $Y$  往往是下降的, 当  $X$  下降的时候,  $Y$  往往是上升的, 这表示两者有较强的负相关性。

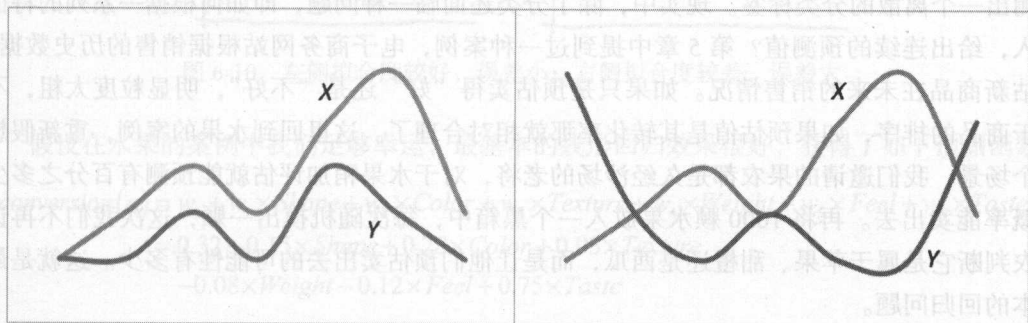


图 6-9 左侧正相关示例; 右侧负相关示例

与欧氏距离一样, 皮尔森系数没有考虑重叠数对结果的影响。计算公式如下:

$$r_p(X, Y) = \frac{1}{n-1} \sum_{i=1}^n \left( \frac{x_i - \bar{X}}{s_x} \right) \times \left( \frac{y_i - \bar{Y}}{s_y} \right)$$

其中  $n$  表示向量维度,  $x_i$  和  $y_i$  分别为两个向量在第  $i$  维的数值。  $\bar{X}$ 、 $\bar{Y}$  分别表示两个向量维度值序列的均值,  $s_x$  和  $s_y$  分别表示两个向量维度值序列的标准差。

□ 斯皮尔曼 (Spearman) 相关系数: 斯皮尔曼等级相关是依据两列成对等级的各对等级数之差来进行计算的, 所以又称为“等级差数法”。这个略有些复杂, 深入解释一下。假设两个向量序列分别为  $X$ 、 $Y$ , 它们的元素个数均为  $N$ , 两个序列的第  $i$  ( $1 \leq i \leq N$ ) 个值分别用  $x_i$ 、 $y_i$  来表示。对  $X$ 、 $Y$  进行排序, 同时为升序或降序, 得到两个排行后的序列  $SX$ 、 $SY$ , 其中元素  $rx_i$ 、 $ry_i$  分别为  $x_i$  在  $X$  中的排行, 以及  $y_i$  在  $Y$  中的排行。将序列  $SX$ 、 $SY$  中的元素对应相减得到一个排行差分集合  $d$ , 其中  $d_i = rx_i - ry_i$ ,  $1 \leq i \leq N$ 。那么  $X$ 、 $Y$  之间的斯皮尔曼等级相关系数可以用如下公式计算得到:

$$r_s(X, Y) = 1 - \frac{6 \times \sum_{i=1}^N d_i^2}{N(N^2 - 1)}$$



直观上来理解，就是两个序列对于其维度排序位置认可的一致程度，如果大多数情况下，在  $SX$  序列中某维度的排行和  $SY$  中该维度的排行非常接近，那么两者的相关度是很高的，相似度也越高。斯皮尔曼取值范围在  $[-1.0, 1.0]$ ，当它们完全一致时为 1.0，完全不一致时为 -1.0。问题是会有大量排序，计算缓慢，不太适合实时性强的计算系统。

至此可见，KNN 算法本身是很容易理解的，但是距离或相似度的计算则需要精心调试和评估，在预测时候的计算量也是相当大的。

### 6.3.2 监督学习——回归

分类问题会根据某个样本中的一系列的特征输入，来最后判定其应该属于哪个分类，预测出一个离散的分类标签。现实中，除了分类还面临一种问题，即如何根据一系列的特征输入，给出连续的预测值？第 5 章中提到过一种案例，电子商务网站根据销售的历史数据，预估新商品在未来的销售情况。如果只是预估卖得“好”还是“不好”，明显粒度太粗，不利于商品的排序，如果预估值是其转化率那就相对合理了。这里回到水果的案例，重新假想一个场景，我们邀请的果农都是久经沙场的老将，对于水果稍加评估就能预测有百分之多少的概率能卖出去。再将 1000 颗水果放入一个黑箱中，每次随机摸出一颗，这次我们不再让果农判断它是属于苹果、甜橙还是西瓜，而是让他们预估卖出去的可能性有多少。这就是最基本的回归问题。

回归的训练和测试流程和分类大体相当，不过采用的具体技术有所不同，它是研究一个或多个随机变量  $y_1, y_2, \dots, y_i$  与另一些变量  $x_1, x_2, \dots, x_k$  之间的关系的统计方法，又称多重回归分析。我们将  $y_1, y_2, \dots, y_i$  称为因变量， $x_1, x_2, \dots, x_k$  称为自变量。通常情况下，因变量的值可以分解为两部分：一部分是由自变量的影响，即表示为自变量相关的函数，其中函数形式已知，可能是线性的也可能是非线性的函数，但包含一些未知参数；另一部分是由于其他未被考虑的因素和随机性的影响，即随机误差。

回归按照不同的维度可以分为以下几种：

□ 按照自变量数量：当自变量  $x$  的个数大于 1 时称为多元回归。

□ 按照因变量数量：当因变量  $y$  的个数大于 1 时称为多重回归。

□ 按照模型：如果因变量和自变量为线性关系时，就称为线性回归模型；如果因变量和自变量为非线性关系时，就称为非线性回归分析模型。举个例子，最简单的情形是一个自变量和一个因变量，且它们大体上有线性关系，称为一元线性回归，即模型为  $Y = a + bX + \varepsilon$ ，这里  $X$  是自变量， $Y$  是因变量， $\varepsilon$  是随机误差，通常假定随机误差的均值为 0。

此处的水果案例就是六元一重回归，六元自变量分别是形状、颜色等 6 个特征维度，一重因变量是最终卖出的概率。至于是否线性回归，需要看训练过程中，线性回归模型是否能很好地拟合学习样本，使得随机误差足够小。如果不能，那就需要尝试非线性的回归模型。图 6-10 展示了二维空间里的拟合程度，图 6-10 中离散的点是训练数据实例，直线是回

归学习后确定的拟合线。从左侧可以看出，实例点和学习后的直线非常接近，误差比较小。而右侧却相反，实例点和学习后的直线距离都比较远。这种情况下我们认为左侧的拟合度要好于右侧，而且左侧学习出的函数参数更可信。而右侧则可能需要考虑换其他非线性的回归函数。

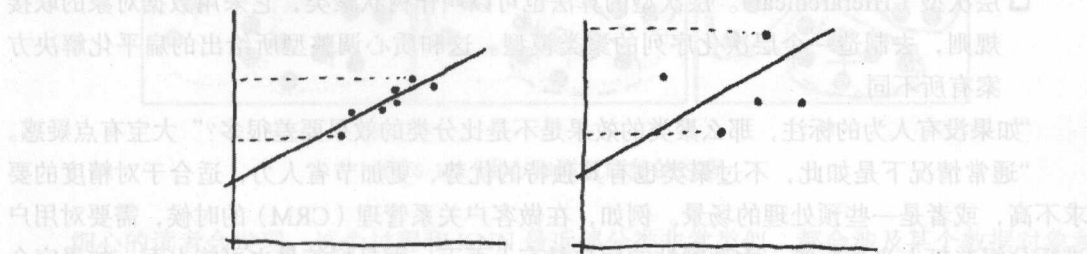


图 6-10 左侧拟合度较好，误差小；右侧拟合度较差，误差大

假设在水果的案例中我们足够幸运，最基本的线形回归效果很好，获得了如下预测函数：

$$\begin{aligned} \text{conversion}(o) &= w_0 + w_1 \times \text{Shape} + w_2 \times \text{Color} + w_3 \times \text{Texture} + w_4 \times \text{Weight} + w_5 \times \text{Feel} + w_6 \times \text{Taste} \\ &= 0.32 + 0.15 \times \text{Shape} + 0.28 \times \text{Color} + 0.03 \times \text{Texture} \\ &\quad - 0.08 \times \text{Weight} - 0.12 \times \text{Feel} + 0.75 \times \text{Taste} \end{aligned}$$

那么，在预测的时候，我们将新的数据对象的各个维度特征值带入上述公式，就可以得到预估的转化率了。

“如果我们将回归的输出离散化处理，那岂不是和分类又一样了？”大宝的脑子里总是充满了怪点子。

“是的，我们完全可以这么做，所以实际生产中回归的技术也常常用于分类。”

### 6.3.3 非监督学习——聚类

综上所述，监督式学习通过训练资料学习建立一个模型，并依此模型推测出新的实例。实际场景中，我们经常会遇到另一种更为原始的情况：不存在任何关于样本的先验知识，而是让我们在没人指导的情形下去将很多东西进行归类。因此，归类系统必须通过一种有效的方法来“发现”样本的内在相似性，然后通过一种被称为“非监督学习”的方法来设计该归类系统。这一节将致力于阐述特征向量的非监督学习方式，通常称为“聚类”（Clustering）。实质上这是一种数据驱动的方法，它试图发现数据自身的内部结构，将数据对象以群组（Cluster）的形式进行分组。假想另外一个场景，还是将 1000 颗水果平摊开，这次我们不会再事先告诉果农，只可能有苹果、甜橙和西瓜三种水果，而是让他们按照水果直接的相似程度，进行归组，更相似的放入一组。这就是最基本的聚类问题了。和分类问题相同的地方在于，果农可能还是会将甜橙和苹果混淆。不同之处在于，在没有修改游戏规则前分类问题下永远只有 3 个分类，而聚类这种方式可能会导致聚出多于或少于 3 个的分组。

根据数据的类型、样本在聚类中的积聚规则，以及应用这些规则所用的方法来看，有很多种聚类算法。大体上可以分为两大类：

□ 质心调整型（Centroid Adjustment）。这种算法使用了一种迭代的方法来调整聚类的典型模式点，也叫聚类的质心，从而形成一系列可以分配给它们的样本。

□ 层次型（Hierarchical）。层次型的算法也可以叫作树状聚类，它采用数据对象的联接规则，去制造一个层次化序列的聚类模型。这和质心调整型所给出的扁平化解决方案有所不同。

“如果没有人为的标注，那么聚类的效果是不是比分类的效果要差很多？”大宝有点疑惑。

“通常情况下是如此，不过聚类也有其独特的优势，更加节省人力，适合于对精度的要求不高，或者是一些预处理的场景。例如，在做客户关系管理（CRM）的时候，需要对用户进行分组并打上兴趣标签。大型网站的用户量有上百万，而且标签量也可能上万，如果完全依赖分类技术对于平台起步而言过于困难。这时可以考虑通过聚类来发现相似的用户，并将他们自动归为一组。然后，对于聚类的初步结果，可以做进一步提炼，对于重要的客户再进行基于分类的标注和处理。”

### 1. K 均值聚类（K-Means Clustering）

K-Means 聚类算法是一种应用最普遍的、通过不断迭代调整  $k$  个聚类质心的算法。这里的质心是群组的中心点，通常用其中成员的平均值来计算。K-Means 是在一个任意多数据集合的基础上，得到一个事先定好群组数量的聚类结果。其中心思想是：最大化总群组内相似度<sup>①</sup>，而群组内相似度是通过群组内各个成员和群组质心比较得到的相似度确定。想法很简单，但是在样本数量达到一定规模后，希望通过排列组合所有的群组划分来找到最大的总群组内相似度几乎是不可能的。于是人们提出如下近似解：

1) 从  $N$  个数据对象中随机选取  $k$  个对象作为质心。因为是第一轮，所以第  $i$  个群组的质心就是选择的第  $i$  个对象，而且只有这一个组员。

2) 对剩余的每个对象测量其和每个质心的相似度，并把它归到最近的质心的群组。

3) 重新计算已经得到的各个群组的质心。这里质心的计算是关键，如果是用特制向量来表示的数据对象，那么最基本的方法是取群组内成员的特制向量，将它们的平均值作为质心的向量表示。

4) 迭代第 2 步和第 3 步，直至新的质心与原质心相等或相差之值小于指定阈值，算法结束。

如果我们将所有的数据对象向量映射到二维空间，图 6-11a、b、c 分别展示了质心和群组逐步调整的过称。a 步骤是第一轮聚类，以及随后计算每个群组的质心。其中的“+”表示质心；b 步骤是第二轮聚类，根据新的质心，计算每个数据点应该属于哪个新的群组；c

① 一般是使用“距离”来表示相似度。这里并没有刻意的限制，KNN 分类算法中介绍了其他非距离表示的相似度。



步骤如此重复, 进入下一轮聚类。

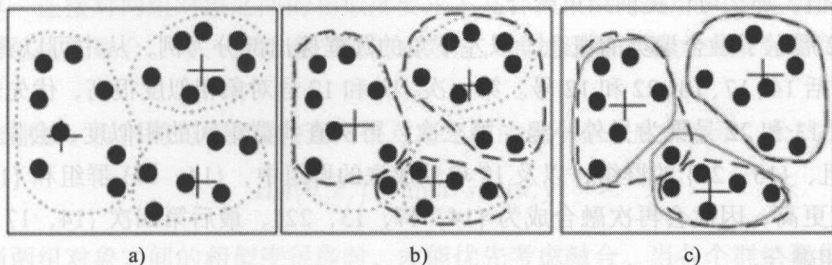


图 6-11 质心和群组调整的过程

细心的读者会发现, 这个过程和 KNN 最近邻分类非常类似, 都会涉及某个数据对象和其他对象或群组质心的相似度计算。最主要的区别在于 KNN 是针对监督式学习的, 训练数据中的分类标签都已经确定, 所以无需多次迭代的优化过程。而 K-Means 中, 一开始的质心和群组选择都是临时的, 在之后的迭代中逐步逼近局部优化的解, 直到达到一个稳定的状态。这也导致在线应用的时候, 相对分类而言, 聚类通常会消耗更多的计算资源和时间。

“小明哥, 这个方法虽然简单易懂, 但是一开始怎样选择这个群组的数量啊, 针对一个新的数据集合,  $k$  值定为多少才比较合适呢? 如果  $k$  值取得太小, 群组可能会切分太细, 每个群组之间的区别不大。如果  $k$  值取得太大, 就怕粒度太粗, 群组内部又差异明显。无论怎样对最后的分析都不利啊。”

“好问题, 这种非监督式的学习确实有一些参数很难得到准确的预估。可以事先在一个较小的数据集合上进行尝试, 然后根据结果和应用场景确定一个经验值。当然, 如果还是不够理想, 可以使用层次型的聚类在一定程度上来缓解这个问题。”

## 2. 层次型聚类 (Hierarchical Clustering)

还有一种类型的聚类方法, 仅仅使用数据对象之间的相似性, 使得同一群组中对象间的相似度, 远远大于不同群组之间的相似度。这就是层次型的聚类。具体又可分为分裂和融合这两种方案。分裂的层次聚类采用自顶向下的策略, 首先是将所有对象置于同一个群组之中, 然后逐渐细分为越来越小的群组, 直到每个对象自成一组, 或者达到了某个阈值条件而终止。融合的层次聚类与分裂的层次聚类相反, 是一种自底向上的策略, 首先将每个对象作为一个群组, 然后将这些原始组合并成越来越大的群组, 直到所有的对象都在一个群组中, 或者达到某个阈值条件而终止。融合的方式在计算上更简单快捷, 因此绝大多数层次聚类的方法都属于这一类, 只是在群组间相似度的定义上有所不同。其大致流程如下:

- 1) 最初给定  $n$  个数据对象, 将每个对象看成一个群组。这样共得到  $n$  个组, 每组仅包含一个对象, 组与组之间的相似度就是它们所包含的对象之间的相似度。
- 2) 找到最接近的两个组并合并成一个, 于是总的组数少了一个。
- 3) 重新计算新的组与所有旧组之间的距离。

4) 重复第 2 步和第 3 步，直到最后合并成一个组为止。如果设置了组数，或者是组间相似度的阈值，那么可以提前结束聚合。

图 6-12 展示了融合聚类的概念。以左下角的圆框标出部分为例，从中可以看到若干数据对象，包括 14、17、13、22 和 12 号。第一次，14 和 17 号对象相似度很高，优先聚为一组，而在第二次 13 和 22 号聚为另外一组。第三次，再次查找群组间的相似度，会发现在 {14, 17} 的群组、{13, 22} 的群组，以及 12 单独成立的群组中，{14, 17} 群组和 {13, 22} 群组的相似度更高，因此会再次融合成为 {14, 17, 13, 22}，最后第四次 {14, 17, 13, 22} 再次和 12 相融合。

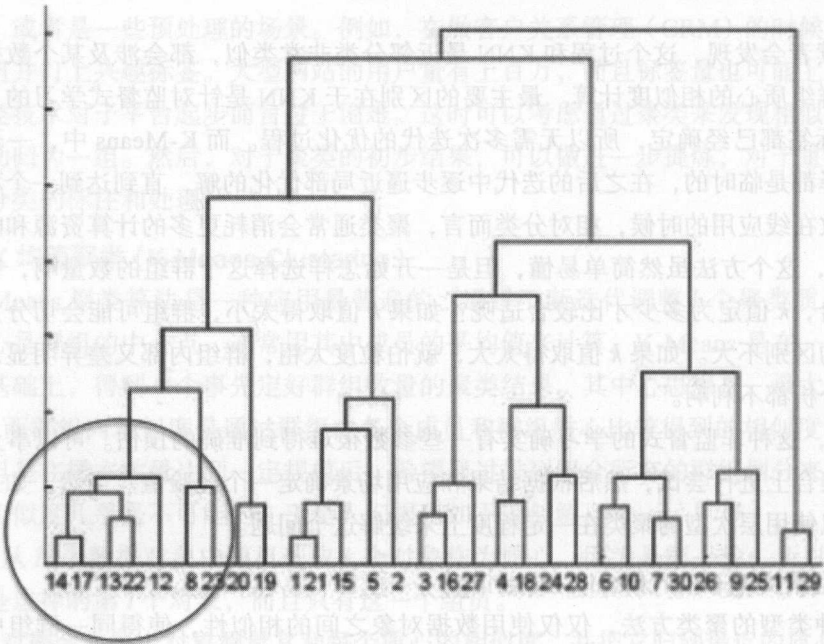


图 6-12 聚类的层次结构

那接下来一个有趣的问题是，如何计算群组之间的相似度呢？之前在 K-Means 聚类中，计算的是单个数据对象和质心间的相似度，就是两个向量间的比较。而现在计算的是两个群组之间的相似度，是两组向量的比较。两组之间的比较工作量肯定更大，常见的方式有三种，分别是单一连接（Single Linkage）、完全连接（Complete Linkage）和平均连接（Average Linkage）。

□ 单一连接：群组间的相似度使用两组对象之间的最大相似度来表示，公式如下。

$$sim(c_i, c_j) = \max_{x \in c_i, y \in c_j} sim(x, y)$$

⊖ 通常 {a, b, c...} 表示这些元素的集合，这里表示它们组成的群组。

其中  $\text{sim}(c_i, c_j)$  表示群组  $i$  和群组  $j$  之间的相似度,  $x$  和  $y$  分别是群组  $i$  和群组  $j$  内的数据对象。单一连接对两组对象之间的相似度要求不高, 只要两个对象间存在较大的相似值就能够使两组优先融合。单一连接会产生链式效应, 通过这种连接方式来融合可以得到丝状结构。

□ 完全连接: 群组间的相似度使用两组对象之间的最小相似度来表示, 公式如下。

$$\text{sim}(c_i, c_j) = \min_{x \in c_i, y \in c_j} \text{sim}(x, y)$$

只有当两组对象之间的相似度很高时, 才能优先考虑融合。当各个群组聚集得比较紧密, 类似球状, 不太符合丝状结构时, 使用单一连接会效果不佳, 这时可以考虑完全连接。

□ 平均连接: 群组间的相似度使用两组对象之间的平均相似度来表示, 公式如下。

$$\text{sim}(c_i, c_j) = \text{avg}_{x \in c_i, y \in c_j} \text{sim}(x, y)$$

相对而言, 这种计算对于各类形状都是比较有效的。

“懂了。这样看来层次型虽然计算量大, 但是对于确定合适的聚类群组还是有所帮助的。另外, 整体感觉, 好像聚类算法比较简单, 也完全不用人工的标注哦, 这样岂不是比分类方便很多?”

“确实不需要人工的分类标注, 节省了很多运营的人力。不过, 通常聚类只能发现数据结构内部的特征, 聚集出来的群组其解释性比不上分类。因此, 聚类比较适合业务需求变化快, 而且对精度要求不高的分析, 例如侦测异常行为、用户分组等。而分类则更适合需求相对稳定、精度要求很高的分析, 例如搜索查询分类、商品目录的分类等。或者, 我们也可以结合这两者, 先用聚类进行初步的分析, 然后再让运营人员通过聚类的结果来构建分类的标注数据。”

## 6.4 挖掘工具

数据挖掘和机器学习的算法正在不断地演变和发展, 也具有一定的技术含量和入门门槛。如果没有较强的数据理论和编程实践的功底, 那么几乎是不可能白手起家来实现具体的某个算法的。那么, 是否存在一些得力的工具来帮助我们完成这些高难度的动作呢? 这里将简单介绍下两个开源的项目, Mahout 和 R。

### 6.4.1 Mahout 简介

第5章提到了最近几年开源的推荐引擎之一——Apache 的 Mahout (<http://mahout.apache.org>)。其实 Mahout 并不限于推荐算法, 还提供了很多分类、聚类和回归挖掘的算法。和其他的挖掘工具和系统相比, Mahout 通过 Apache Hadoop 将算法有效地扩展到分布式系统中。随着训练数据的不断增加, 非分布式的系统用于训练的时间或硬件需求并不是线性增



加的，这点已经在计算机系统中被广泛验证。假设 5 倍的训练数据会导致 100 倍的训练时间，那将是用户无法接受的事实。Mahout 可以将数据切分成小块，并通过 Hadoop 的 HDFS 存储，并通过 Map-Reduce 来计算。分布式的协调处理将时间消耗尽量控制在线性范围<sup>①</sup>。因此，当训练数据量非常庞大的时候，Mahout 的优势就会体现出来。按照其官方的说法就是，这个规模的临界点在百万到千万级，具体还要看每个数据对象和挖掘模型的复杂程度。

Mahout 中的分类算法，除了前面介绍的决策树、朴素贝叶斯和回归，还包括了支持向量机（Support Vector Machine）、随机森林（Random Forests）、神经网络（Neural Network）和隐马尔科夫模型（Hidden Markov Model），等等。支持向量机属于一般化线性分类器，特点是能够同时最小化经验误差和最大化几何边缘区。随机森林是一个包含多个决策树的分类器，在决策树的基础上衍生而来，其分类标签的输出由多个决策树的输出投票来决定，这在一定程度上弥补了单个决策树的缺点。最近几年随着深度学习（Deep Learning）的流行，神经网络再次受到人们的密切关注。众所周知，人脑是一个高度复杂的、非线性的并行处理系统。人工建立的神经网络起源于对生物神经元的研究，并试图模拟人脑的思维方式，对数据进行分类、预测及聚类。隐马尔科夫模型更适合于有序列特性的数据挖掘，例如语音识别、手写识别和自然语音处理等，其中文字和笔画的出现顺序都会对后面的预测大有帮助。

Mahout 中的聚类算法同样实现了 K-Means 算法，以及针对其所做的优化和扩展。对于 K-Means，Mahout 提供了基本的基于内存的实现和基于 Hadoop 的 Map/Reduce 的实现。不过 K-Means 的参数选择也有很明显的問題，首先 K-Means 需要在执行聚类之前就要有明确的群组个数设置，但在处理大部分问题时，这一点都是很难事先知道的，一般需要通过多次试验才能找出一个最优的  $k$  值；此外，由于算法在最开始采用的是随机选择初始质心的方法，所以算法对噪音和异常点的容忍能力较差。一旦噪音点在一开始就被选作群组的初始质心，那它就会对后面的整个聚类过程带来很大的负面影响。因此，Mahout 还实现了 Canopy 算法，用于优化 K-Means 聚类的效果。Canopy 聚类算法的基本原则是：首先应用低成本的近似的相似度计算方法将数据高效地分为多个“Canopy”，Canopy 之间可以有重叠的部分，然后采用严格的距离计算方式准确地计算在同一 Canopy 中的点，并将它们分配到最合适的群组中。Canopy 聚类算法经常用作 K-Means 聚类算法的预处理，用来找到合适的  $k$  值和群组质心。

Mahout 中实现的另一个扩展是模糊 K-Means 聚类，它的基本原理和 K-Means 一样，只是它的聚类结果允许存在某个数据对象属于多个群组的情况，属于可重叠聚类。与 K-Means 的原理类似，模糊 K-Means 也是在待聚类对象向量集合上循环，计算给定向量与各个群组的相似性，然后通过参数设定对象最多属于几个群组。

Mahout 除了 K-Means 相关的三种聚类算法外，还实现了一个基于概率分布模型的聚类算法，狄利克雷聚类（Dirichlet Processes Clustering）。其原理是定义一个分布模型（简单的

<sup>①</sup> 分布式系统有一些额外的消耗用于通信和协调，例如在网络中传输数据，因此无法保证资源被 100% 地利用。

例如圆形、三角形等，复杂的例如正则分布、泊松分布等)，然后按照模型对数据进行分类，将不同的对象加入一个模型，模型会增长或收缩；每一轮过后需要对模型的各个参数进行重新计算，同时估计对象属于这个模型的概率。所以说，基于模型的聚类算法的核心是定义模型，对于一个聚类问题，模型定义的优劣会直接影响聚类的结果。

“总体而言，Mahout 提供了较为丰富的算法库，待熟悉之后你也可以自己优化具体实现，是算法编程入门不错的选择。”

“好的，小明哥，我会深入研究一下的。”

## 6.4.2 R 简介

Mahout 虽然很强大，但是有个很明显的问题：需要使用者懂得复杂的编程语言设计和测试，例如 Java，而这对于不少读者而言门槛实在是太高了。那有没有一个工具可以在不具备过多计算机编程知识的前提下使用呢？R (<https://www.r-project.org/>) 提供了一套解决方案。R 虽然只有一个字母，但是它代表了一整套解决方案，包括 R 语言及其对应的系统工具。R 语言是统计领域广泛使用的诞生于 1980 年左右的 S 语言的一个分支。S 语言是由 AT&T 贝尔实验室开发的一种用来进行数据探索、统计分析和作图的解释型语言，R 语言可以认为是 S 语言的一种实现。相对于 Java 和 Mahout 而言，R 的脚本式语言更容易被理解，而且它还提供了颇为丰富的范例供使用者直接使用。此外 R 的交互式环境和可视化工具也极大地提高了生产效率，人们可以从广泛的来源获取数据，将数据整合到一起，并对其进行清洗等预处理，然后用不同的模型和方法进行分析，最后将结果通过直观的、可视化的方式展现出来。当然，还有一点非常关键：R 也是免费的，相对于价格不菲的商业软件而言，它的性价比实在是太高了。下面来介绍一下 R 的几个主要功能。

### 1. 交互式的环境

目前为止，R 的最新版本是 3.2.3，可以在 <https://cran.r-project.org/mirrors.html> 上选择合适的平台下载并安装。安装后运行，你将看到如图 6-13 所示的界面，实际上就是一个输入命令的终端。你可以在提示符“>”后面输入并执行一条命令，或者通过书写脚本一次性执行多个命令。R 支持多种数据类型，如向量、矩阵、列表等。让我们从最简单的函数 c() 开始，它可以让你输入一个向量，例如下面的这两条命令：

```
> apple.a <- c(1,1,1,2,1,1)
> apple.a
[1] 1 1 1 2 1 1
```

灵感依旧来自水果的案例，第一条命令“> apple.a <- c(1,1,1,2,1,1)”是将苹果 a 的特征值以向量的形式赋予对象 apple.a，其中“<-”表示赋值。第二条命令是显示 apple.a，是不是很简单呢？以此类推，可以手动建立表 6-3 中所有的水果对象，展示如下：

```
> apple.b <- c(1,1,1,1,1,1)
```

```

> apple.c <- c(2,3,1,1,2,1)
> orange.a <- c(2,2,1,1,2,2)
> orange.b <- c(2,2,1,2,2,2)
> orange.c <- c(1,2,1,2,1,1)
> watermelon.a <- c(3,3,2,3,1,2)
> watermelon.b <- c(3,3,2,3,1,1)
> watermelon.c <- c(3,3,2,3,1,2)
> watermelon.d <- c(1,3,2,3,2,2)
> ls()
[1] "apple.a"      "apple.b"      "apple.c"      "applea"      "orange.a"
[6] "orange.b"     "orange.c"     "watermelon.a" "watermelon.b" "watermelon.c"
[11] "watermelon.d"

```

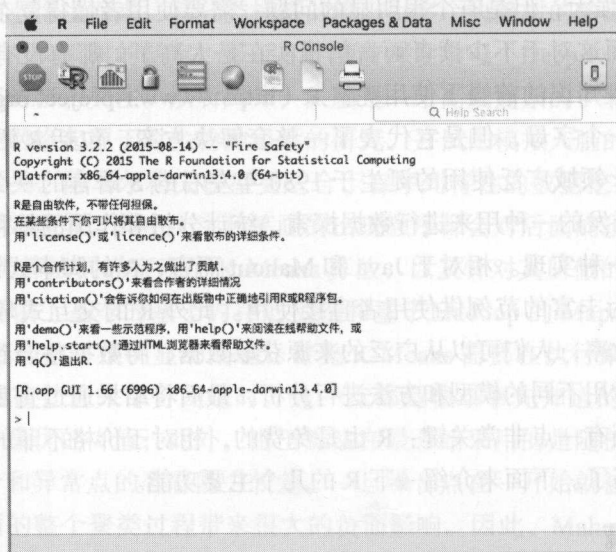


图 6-13 R 启动后的初始画面，显示了 R 的版本和帮助信息

其中 `ls()` 是列出当前定义的所有对象。除了允许用户在终端手工输入信息，R 还支持从文本文件、数据库系统，甚至是其他统计软件上导入数据，对于数据源的整合很有益处。有了这些数据，要进行基础的处理也非常快捷。下面的命令分别列出了西瓜 a 作为数组的最大值、最小值、均值、中位数、方差和标准差的数值。

```

> max(watermelon.a)
[1] 3
> min(watermelon.a)
[1] 1
> mean(watermelon.a)
[1] 2.333333
> median(watermelon.a)
[1] 2.5
> var(watermelon.a)

```



```
[1] 0.6666667
> sd(watermelon.a)
[1] 0.8164966
```

至此，你已经开始了 R 工作的第一步，那么如何保存这些成果呢？别急，R 还提供了工作间（Workspace）的概念，即指当前的工作环境。通过保存工作间的镜像，你可以存储用户定义的数据对象和一些设置，R 在下次启动时会自动加载所有的这些内容。

## 2. 丰富的包 (Package)

R 提供了大量开箱即用的功能，称为包。你可以将其理解为 R 社区用户贡献的模块，从简单的数据处理，到复杂的数据挖掘和机器学习算法，都有所涵盖。截至本书撰写的时候，包的总数已经超过了 7000 个，横跨多个领域。初次安装 R 的时候会自带一系列默认的包，提供默认的函数和数据集，其他的扩展可根据需要下载并安装。

## 3. 直观的图示化

俗话说，“一图胜千言”，图形展示是最高效且最形象的描述手段，巧妙的图像展示也是高质量数据分析报告的必备内容。因此一款优秀的统计分析软件必须具备强大的图形展示功能，R 也不例外。同样，画图都是有现成的函数可供调用，包括直方图（hist()）、散点图（plot()）、柱状图（barplot()）、饼图（pie()）、箱线图（boxplot()）、星相图（stars()）、脸谱图（faces()）、茎叶图（stem()）等。

接下来，我们来看看通过 R 实现数据挖掘的几个例子。第一个关于分类，我们使用 R 自带的鸢尾花（iris）数据集来做来做决策树分析。这个数据集中一共定义了这种花的 3 个类型：setosa、versicolor 和 virginica。下面通过输入命令 iris 来展示整个数据集，在 R 中它是通过数据框架（Data Frame）来加载的：

```
> iris
      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1             5.1         3.5          1.4          0.2     setosa
2             4.9         3.0          1.4          0.2     setosa
3             4.7         3.2          1.3          0.2     setosa
...
51            7.0         3.2          4.7          1.4 versicolor
52            6.4         3.2          4.5          1.5 versicolor
53            6.9         3.1          4.9          1.5 versicolor
...
101           6.3         3.3          6.0          2.5  virginica
102           5.8         2.7          5.1          1.9  virginica
103           7.1         3.0          5.9          2.1  virginica
```

在有了 iris 的数据作为训练样本后，还需要导入 rpart 包（这个包可能需要额外下载）来做决策树分析，命令如下：

```
> install.packages("rpart")
> library(rpart)
```

rpart 包的决策树处理首先对所有的特征值和所有的分割点进行评估，最佳的选择是使分割后组内的数据更为“一致”，这种一致性的默认度量是基尼系数（Gini），可以达到和信息增益类似的效果。有了 rpart 包，决策树分析只需要 1 行命令即可：

```
> iris.rp=rpart(formula=Species~.,data=iris,method="class")
```

rpart() 是构建回归树的函数，其中 formula 回归方程形式，或者说学习的目标函数，这里放的是分类标签 Species。data 包含前面方程中特征变量的数据框，这里放的是 iris。method 则根据树末端的数据类型选择相应的变量分割方法，分为连续型“anova”、离散型“class”、计数型“poisson”和生存分析型“exp”。运行结束后，再次展示存放在 iris.rp 中的结果：

```
> iris.rp
n= 150
node), split, n, loss, yval, (yprob)
* denotes terminal node
1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
2) Petal.Length< 2.45 50 0 setosa (1.00000000 0.00000000 0.00000000) *
3) Petal.Length>=2.45 100 50 versicolor (0.00000000 0.50000000 0.50000000)
6) Petal.Width< 1.75 54 5 versicolor (0.00000000 0.90740741 0.09259259) *
7) Petal.Width>=1.75 46 1 virginica (0.00000000 0.02173913 0.97826087) *
```

细心的你可能会发现这就是一棵树，不过通过文字表达实在是晦涩难懂。我们来利用 R 强大的画图功能吧，代码如下：

```
> plot(iris.rp,uniform=T,branch=0,margin=0.01,main="DecisionTree")
> text(iris.rp,use.n=T,fancy=T,col="red")
```

其中 plot() 是画线函数，而 text() 是在图像上打标签。在 plot() 中 branch 控制分支线的形状，0 表示 V 形分支，而 1 则表示方肩分支。uniform 为 T (True) 则表示节点的垂直间距是一致的，如果其值设置为 F (False)，那么节点隔开的间距就按照子节点的比例来设置，这样可能会导致标签看不清楚。另外，margin 是离边框的距离，main 是图像的主标题。在 text() 中，use.n 为 T(True) 则表示在标签里显示分组内各个分类的实例数量，fancy 为 T(True) 时会添加椭圆和方形的框以用于美化，而 col 则确定了标签字符的颜色。最后，我们可以得到如图 6-14 所示的决策树图像。

最后，来看下 R 中的层次型聚类是如何进行的。我们将表 6-3 中的 10 颗水果的数据导入：

```
> apple.a <- c(1,1,1,2,1,1,"apple")
> apple.b <- c(1,1,1,1,1,1,"apple")
> apple.c <- c(2,3,1,1,2,1,"apple")
```

```
> orange.a <- c(2,2,1,1,2,2,"orange")
> orange.b <- c(2,2,1,2,2,2,"orange")
> orange.c <- c(1,2,1,2,1,1,"orange")
> watermelon.a <- c(3,3,2,3,1,2,"watermelon")
> watermelon.b <- c(3,3,2,3,1,1,"watermelon")
> watermelon.c <- c(3,3,2,3,1,2,"watermelon")
> watermelon.d <- c(1,3,2,3,2,2,"watermelon")
```

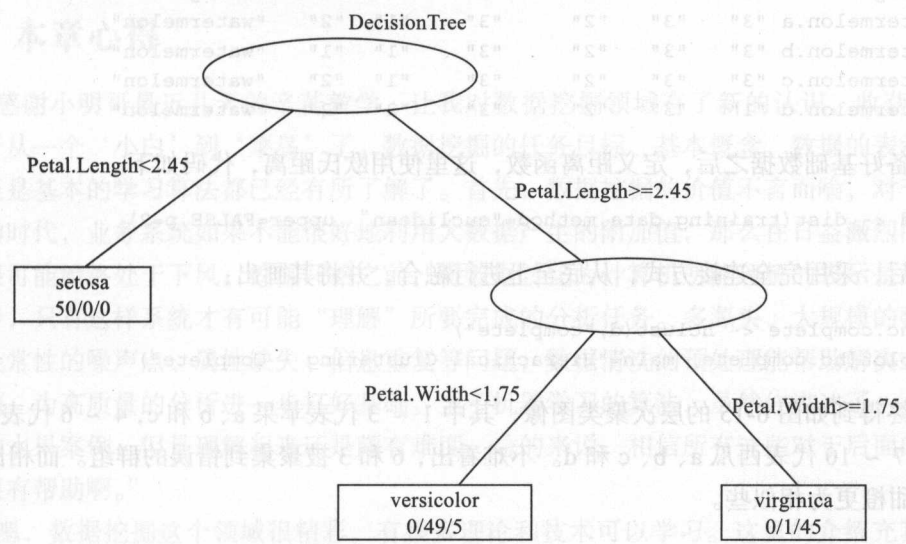


图 6-14 iris 数据集生成的决策树

然后生成训练数据的矩阵:

```
> training.data <- rbind(apple.a, apple.b, apple.c, orange.a, orange.b,
orange.c, watermelon.a, watermelon.b, watermelon.c, watermelon.d)
> training.data
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
apple.a	"1"	"1"	"1"	"2"	"1"	"1"	"apple"
apple.b	"1"	"1"	"1"	"1"	"1"	"1"	"apple"
apple.c	"2"	"3"	"1"	"1"	"2"	"1"	"apple"
orange.a	"2"	"2"	"1"	"1"	"2"	"2"	"orange"
orange.b	"2"	"2"	"1"	"2"	"2"	"2"	"orange"
orange.c	"1"	"2"	"1"	"2"	"1"	"1"	"orange"
watermelon.a	"3"	"3"	"2"	"3"	"1"	"2"	"watermelon"
watermelon.b	"3"	"3"	"2"	"3"	"1"	"1"	"watermelon"
watermelon.c	"3"	"3"	"2"	"3"	"1"	"2"	"watermelon"
watermelon.d	"1"	"3"	"2"	"3"	"2"	"2"	"watermelon"

加上特征列的名称以提升可读性:

```
> colnames(training.data) <- c("Shape", "Color", "Texture", "Weight", "Feel",
```



```
"Taste", "Class")
```

```
> training.data
```

	Shape	Color	Texture	Weight	Feel	Taste	Class
apple.a	"1"	"1"	"1"	"2"	"1"	"1"	"apple"
apple.b	"1"	"1"	"1"	"1"	"1"	"1"	"apple"
apple.c	"2"	"3"	"1"	"1"	"2"	"1"	"apple"
orange.a	"2"	"2"	"1"	"1"	"2"	"2"	"orange"
orange.b	"2"	"2"	"1"	"2"	"2"	"2"	"orange"
orange.c	"1"	"2"	"1"	"2"	"1"	"1"	"orange"
watermelon.a	"3"	"3"	"2"	"3"	"1"	"2"	"watermelon"
watermelon.b	"3"	"3"	"2"	"3"	"1"	"1"	"watermelon"
watermelon.c	"3"	"3"	"2"	"3"	"1"	"2"	"watermelon"
watermelon.d	"1"	"3"	"2"	"3"	"2"	"2"	"watermelon"

准备好基础数据之后，定义距离函数，这里使用欧氏距离，代码如下。

```
> d <- dist(training.data,method="euclidean",upper=FALSE,p=2)
```

最后，采用完全连接方式，从底至上进行融合，并将其画出：

```
> hc.complete <- hclust(d,"complete")
```

```
> plot(hc.complete, main="Hierarchical Clustering - Complete")
```

最终得到如图 6-15 的层次聚类图像，其中 1 ~ 3 代表苹果 a、b 和 c，4 ~ 6 代表甜橙 a、b 和 c，7 ~ 10 代表西瓜 a、b、c 和 d。不难看出，6 和 3 被聚集到错误的群组。而相比西瓜，苹果和甜橙更为相似些。

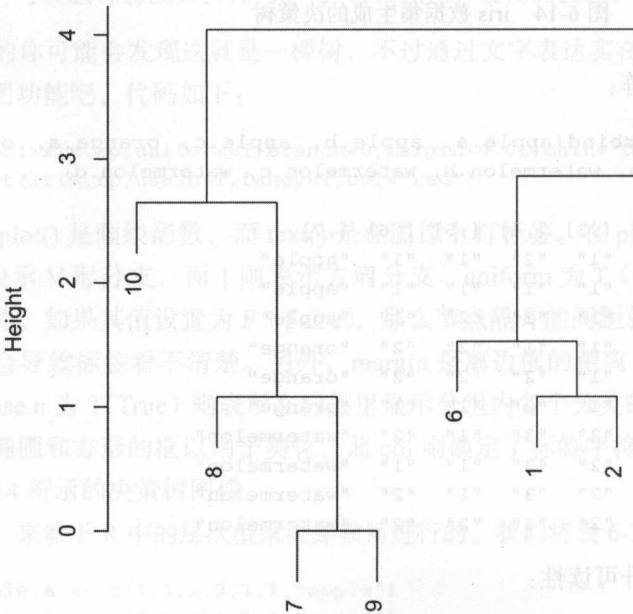


图 6-15 水果数据集生成的层次聚类

“哇，R 的函数和功能真的很强大啊，这些复杂的算法都是几句命令就可以搞定。那么我们可以在 R 里实现自己的算法和函数吗？”

“当然可以，7000 多的包就是社区同仁们不断努力扩展的结果。此外，R 的强大之处远远不限于其本身。我们甚至可以在 Hadoop 的环境中使用 R，或者是将 R 语言和 Java 语言结合使用。”

## 6.5 本章心得

“感谢小明哥最近几天的辛苦教学，让我对数据挖掘领域有了新的认识，收获颇丰啊，我终于从一个‘小白’到‘菜鸟’了。数据挖掘的任务目标、基本概念、数据的表示和预处理甚至是基本的学习算法都已经有所了解。首先，数据挖掘的价值不言而喻，对于大数据驱动的时代，业务系统如果不能很好地利用大数据产生的附加值，那么在日益激烈的商业竞争中很可能始终处于下风。挖掘开始之前，将数据转化为计算机所能理解的表示是最重要的第一步，只有这样系统才有可能“理解”所要完成的分析任务。多源头、大规模的数据也会导致经常性的噪声点、属性缺失、信息重复等问题，数据清洗的预处理能帮助解决或缓解这些问题，为高质量的分析进一步打好基础。至于机器学习的算法，虽然你讲述了一个非常接地气的水果案例，但是理解起来还是颇有难度。总的来说，相信所有这些对于后面的实际应用会很有帮助啊。”

“嗯，数据挖掘这个领域很精彩，有很多理论和技术可以学习。这里的介绍充其量只能算‘师傅领进门’，还有很广阔的天地等你来探索。特别是算法部分，正如你所说的博大精深，不仅难度较高，还有些未介绍的如神经网络、频繁项挖掘、基于图论的挖掘等，还需要不懈的坚持和努力哦。”

“好的，不过，我还是想先磨练下分类、聚类和回归，相信对我们创业项目的实际开发会很有帮助。比如，我想到了利用分类，建立一个自动的商品发布系统，大幅减轻运营人员的工作量。当商品销售累积了一定的历史数据后，我们还可以通过回归来帮助商家预测销售额和转化率，提升我们的商品排序。至于聚类技术，应该会对侦测重复铺货起到作用。”

## 6.6 参考资料

“不错，从你的心得体会可以看出你很用心啊，不仅关注技术本身，还考虑到了现实中的应用开发。无论如何，数据挖掘和信息检索一样，技术含量都是比较高的，这里给你一些可以深入阅读的参考书。”

□《数据挖掘：概念与技术（原书第3版）》

作者：韩家炜，Micheline Kamber，裴健

译者：范明，孟小峰

本书介绍了数据挖掘的发展历史、基本概念、主要流程和主流算法，新版重新组织了全书的技术内容，并重点论述了数据预处理、频繁模式挖掘、分类和聚类等内容。概念部分适合入门阅读，技术部分有较高的理论深度，适合专业人士的阅读。

#### □《机器学习》

作者：Mitchell T.M.

译者：曾华军，张银奎等

本书是较早的机器学习入门教材，深入浅出地讲述了机器学习的背景和概念，以及常用的算法类型，适合初学者阅读。

#### □《机器学习导论（原书第2版）》

作者：Ethem Alpaydin

译者：范明，咎红英，牛常勇

本书是最近出版的机器学习书籍，内容相对较新，不过难度也高一些，需要一定的数据或统计学功底。

#### □《机器学习实战》

作者：Peter Harrington

译者：李锐，李鹏，曲亚东，王斌

通过 Python 语言实现了不少常用挖掘算法的样例，实践性很强，适合工程开发人员。

#### □《Mahout 实战》

作者：Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman

译者：王斌，韩冀中，万吉

如第5章所述，本书重在“实战”，介绍了 Mahout 对推荐系统、聚类和分类算法的支持。

#### □《R 语言实战》

作者：Robert I. Kabacoff

译者：高涛，肖楠，陈钢

作为 R 语言及其工具的入门图书，是相对不错的选择，本书介绍了 R 的基本概念和工具的运用。



## 第7章 Chapter 7

# 效能评估

时间一晃而过，自小明向大宝介绍完数据挖掘后，两个月过去了。“这小子最近在忙些啥呢，一点消息也没有”，小明心里一直在嘀咕。巧的很，这天他们在转角的咖啡馆不期而遇。

“嗨，大宝，这么巧！”

“嗨，小明哥，真巧。你也来喝咖啡？”

“是啊，周末在家感觉无精打采的，来这边喝杯猫屎咖啡提提神。对了，在看什么啊，咖啡馆里也这么认真。”

“哦，正好这里可以上网，在网上查一些数据挖掘的算法介绍呢。”

“挺认真的嘛，加油加油！”

“哪里哪里，刚刚入门。不过，我最近看了很多介绍，也动手做了一些尝试，慢慢地心里产生了一个疑问：这么多方法和算法，到底哪个效果最好呢？如果采用了新的解决方案，应该如何衡量才能得知是否有进步呢？”

“嗯……这确实是个好问题。很难说哪个方法就一定更好，不同的数据集合、不同的应用场景、不同的访问流量都可能导致完全不同的结论。不过，还是有一些基本的准则可以帮助我们衡量方法的有效性。”

“那快讲讲，小弟洗耳恭听啊！”

“我觉得整体上可以称为效能评估……”

效能，在百度百科上最基本的解释是达到系统目标的程度，或者是系统期望达到一组具体任务要求的程度。在这里，将介绍它的另外一层含义：效能 = 效果 + 性能。效果，就是指对于我们的任务而言，相关度、准确度等最终能有多大程度的满足，这 and 具体的数据处理应用是紧密关联在一起的。例如，可能信息检索里的效果评估和数据挖掘里的就有所不

同。而性能，就是指系统对用户请求的时间、速度等有多大程度的满足，一般都是系统级指标，对于不同的应用领域来说都是通用的。为了便于理解，让我们回顾下第6章关于水果的例子：在区分苹果、甜橙和西瓜的时候，之前采用的是决策树模型。有一天，有人提议尝试一下朴素贝叶斯。那么，你可能会问他/她：“这个新的分类技术的实现，相比之前的，会好在哪里？是分类时更准确了，还是分类速度更快速了？如何给出尽量客观的结论？”我们可以认为是否“更准确”关乎这里所说的效果，而是否“更快速”关乎这里所说的性能。效果和性能是最基本也是最关键的两大衡量指标。本章会按照不同的数据处理和挖掘方法，分门别类地进行阐述、归纳和总结。7.1节会介绍效果评估的大体方法有哪些，以及在不同应用中常见的衡量指标。7.2节会介绍衡量性能的指标，包括基于理论的算法复杂度和基于应用的系统指标。

## 7.1 效果评估

像信息检索和数据挖掘这样高级的数据处理，已经不再是简单的计数或是加和。不得不承认，找到用户觉得相关的文章，或者是将产品进行正确的分类都是很有挑战性的工作。某些时候，即使是人类也不一定能100%准确地完成。这种情况下，使用更为科学、尽量客观的方式来测量实际效果就非常有价值了。因此“评估”本身也变成了一门学问，如果从评测的整体流程来细分，主要有三种方式。

❑ 离线评估：在系统没有上线之前，使用现有的标注数据集合来评估。其优势在于，上线之前的测试便于设计者发现问题。万一发现有可以改进之处，技术调整后也可以再次进行评估，反复测试的效率非常之高。其问题在于需要运营人员付出大量精力来标注数据，以作为标准答案。不过，对于监督学习而言，由于其本身就需要大量人为标注作为训练样本，因此可能无需额外的工作量，离线的方式非常适合它。

❑ 用户访谈：在系统大规模推广前，邀请一小部分用户使用系统来完成指定的任务，然后回答关于用户体验的问题。这个阶段，并不要求系统一定离线或在线，只需要保证受邀的用户群可以正常试用即可。相对于离线测试而言，用户访谈允许更多的交互，可以更深入地理解用户的反馈。当然，该方法同样面临人力的问题，问卷的设计、用户的使用和反馈，以及体验的总结都是十分耗费精力的。此外，很多时候用户的体验过程无法复制，系统替换后很可能需要重新访谈，在这点上用户访谈不如离线评估。

❑ 在线评估：通过已经在线部署的系统，大量记录用户的行为，然后根据反馈数据直接做评估。该方法可以认为是用户访谈的一种大规模扩张，不过这种情形下被记录的用户往往不知道测试的存在，而且我们也很难获取行为背后用户的真实想法，只能做一些大胆的假设，因此测试的结果需要深入的分析。在线方法的最大优势莫过于节省人力，如果在线流量足够，甚至可以同时评估多个技术方案，效率极高。现

实环境中，AB 测试（AB Testing）是在线评估最常见的实现形式。

7.1.1 离线评估

离线评估是通过事先收集的用户选择或打分数据来进行测试，使用这种数据集合可以尝试模拟用户与系统的交互行为。这里以最简单的二值相关性判定为例，其要素包括：

- 一个预先选择的数据对象集合  $D$ 。
- 一个用于测试的信息需求集合  $I$ 。
- 一组相关性判定结果，对每个需求  $I$  和数据  $D$  的二元组  $\langle i_k, d_l \rangle$  对而言，赋予一个二值判断结果：0 或 1。其中  $i_k$  属于集合  $I$ ， $d_l$  属于集合  $D$ ，0 表示  $i_k$  和  $d_l$  不相关，1 表示  $i_k$  和  $d_l$  相关。这也是信息检索系统最常规的系统评价方法。

如此，数据集中关于相关或不相关的判定，称为绝对真理（Ground Truth）或黄金标准（Gold Standard），可以将其简单地比作考试的标准答案。尽管如此，值得注意的是，这种方式有一个很强的假设：无论是系统需要处理的数据，还是用户使用系统的行为，在上线前后都应该尽量保持一致。试想，如果离线时处理的数据分布和线上相去甚远，那么训练的模型对于待处理的新数据而言就不具代表性了，预测的效果也会不好。这种情况也可以称为训练“过拟合”，表示过于迎合离线的标注样本，而不能很好地处理新的数据。另一方面，如果离线评测人员的标准，和即将使用在线系统的人员有所不同，很显然同样会影响离线评估的效果。如果你的应用基本能满足这个假设，那么离线的方式还是非常适用的，可以花费相对较小的人力，高效率地测试大量的算法和模型。此外，离线评估还有助于不同的领域间共享已有的标注数据，逐步形成更为成熟的整体评测方案。下面就分别介绍离线方式中常用的信息检索评测指标、数据挖掘评测指标和交叉验证的方法。

1. 信息检索的评估

检索质量最基本的两个评测指标是精度（Precision）<sup>①</sup>和召回率（Recall）。假设在一个数据集  $D$  中，与一个信息需求  $i_k$  相关的数据集合是  $m$ ，在用户输入需求后，某个检索系统返回了结果集合  $n$ ，而  $o$  是集合  $m$  和  $n$  的交集，具体关系如图 7-1 所示。

那么精度  $p$  的定义为：

$$p = \frac{|o|}{|n|} \quad p \in [0,1]$$

召回率  $r$  的定义为：

$$r = \frac{|o|}{|m|} \quad r \in [0,1]$$

① 也有文献将 Precision 翻译为准确率。这里译为精度是为了与后面分类中的准确率（Accuracy）进行区别。



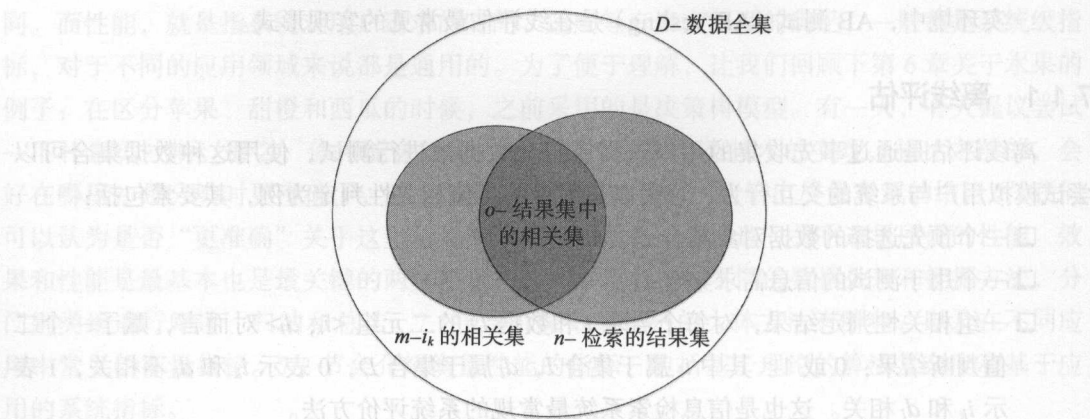


图 7-1 用于定义精度和召回率的三个集合

上述的定义是假设用户已经检测了整个返回的结果集合  $n$ 。试想一下真实的场景，用户最典型的行为是从头到尾开始阅读，因此随着检阅数据的不断增加，精度和召回率是在不断发生变化的。这里结合第 5 章中提到的美食的案例来阐述，表 7-1 展示了拥有 10 篇文章的文档集合。后面标明了每篇文章是否包含“美食”这个关键词，以及是否与美食的主题相关<sup>①</sup>，可以看到，总共有 8 篇文章包含关键词“美食”，还有 5 篇是和美食的主题相关。

“小明哥，这里我能提个问题吗？”

“当然，请讲。”

“如何判断文章和某个主题是否相关呢？每个人也许都会有不同的观点吧。”

“一点也不错，相关性的判定总是带有主观性，这也是离线测试面临的问题。在实际应用中比较像电视里播放的选秀节目，需要专业的人士来做裁判，而且可能需要多个评委来综合评定，避免个别人的主观想法影响了整个测试的集合。这里让我们先来假设表 7-1 中的判断都是准确的吧。”

表 7-1 美食的文档集合示例

文章 ID	标题	是否包含关键词“美食”	是否和美食相关
1	最上瘾的绝味川菜	是	是
2	爱心歌手出席现场活动	是	否
3	美丽的摩登都市：上海	否	否
4	苏州园林赏析	否	否
5	舌尖上的历史	是	是
6	中国国家地理	否	否
7	2015 娱乐风向标	否	否
8	在家吃遍八大菜系	是	是

① 包含 1 个关键词并不代表一定与这个主题相关，在第 5 章的检索模型里有所介绍。

(续)

文章 ID	标题	是否包含关键词“美食”	是否和美食相关
9	新片速递	否	否
10	居家生活好帮手	是	否
11	北京美食地图	是	是
12	二战军事大事件	否	否
13	澳大利亚旅游指南	是	否
14	世界饮食文化	是	是
15	电子竞技发展现状	否	否

当用户搜索“美食”这个关键词时，某系统 A 将按如下形式依次返回 8 篇包含“美食”关键词的文章。

文章 ID	标题	是否相关
5	舌尖上的历史	是
10	居家生活好帮手	否
11	北京美食地图	是
14	世界饮食文化	是
2	爱心歌手出席现场活动	否
1	最上瘾的绝味川菜	是
8	在家吃遍八大菜系	是
13	澳大利亚旅游指南	否

假设用户从第一个排位开始阅读，直到读完全部 8 个返回的结果。看到第一位的文章 5，属于相关，那么这个时候的精度是  $1/1 = 100\%$ ，其中分子 1 和分母 1 都表示文章 5，召回率是  $1/5 = 20\%$ ，其中分子 1 表示文章 5，分母 5 表示文章 1、5、8、11 和 14。再往下，看到第二位的文章 10，不相关，那么这个时候的精度是  $1/2 = 50\%$ ，其中分母 2 表示前两位的文章 5 和 10，召回率仍然是  $1/5 = 20\%$ 。以此类推，看到第 3 ~ 8 位后，精度分别是 66.67%、75%、60%、66.67%、71.43%、62.5%，而召回率分别是 40%、60%、60%、80%、100%、100%。以精度和召回率为两个轴线，我们可以画出如图 7-2 所示的曲线图，其中黑色的直线表示趋势线。

假设另一个系统 B 有不同的返回，返回结果如下。

文章 ID	标题	是否相关
5	舌尖上的历史	是
8	在家吃遍八大菜系	是
14	世界饮食文化	是
2	爱心歌手出席现场活动	否
1	最上瘾的绝味川菜	是

10	居家生活好帮手	否
11	北京美食地图	是
13	澳大利亚旅游指南	否

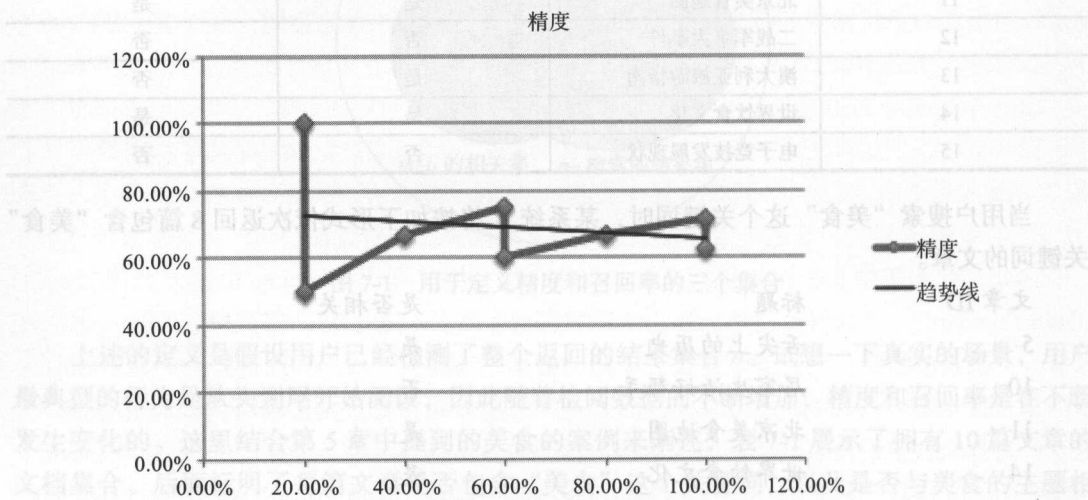


图 7-2 精度和召回率的对应关系

那么，用户从第一个排位开始阅读，直到读完全部 8 个返回的结果，其精准度依次为 100%、100%、100%、75%、80%、66.67%、71.43%、62.5%，而召回率分别是 20%、40%、60%、60%、80%、80%、100%、100%。结合图 7-2，可以对比系统 A 和 B 产生的 2 条曲线图，如图 7-3 所示，其中黑色直线和黑色虚线分别表示 2 条趋势线。

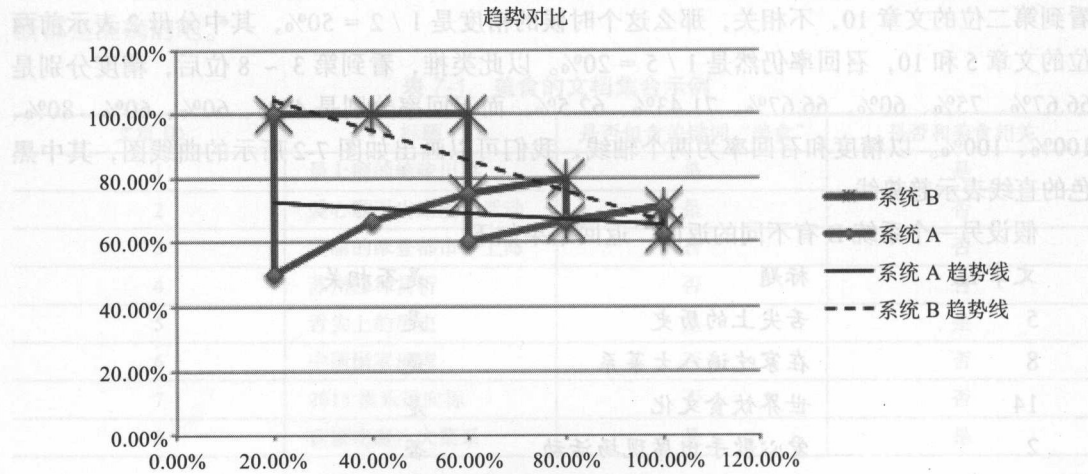


图 7-3 系统 A、B 精度和召回率曲线对比



从图 7-2 和图 7-3 中可以发现两点:

- 随着返回结果数量的增加, 召回率呈现的是逐步上升的趋势, 而精度虽有波动, 但整体上是下降趋势。召回率和精度大体上是呈现反比关系的。这也是实际应用中常见的模式, 而且是检索质量越高的系统, 这个特征就越明显。可见, 召回率和精度虽然都很重要, 但是鱼和熊掌不可兼得。因此, 设计者应该根据实际需求, 尽量均衡这两者之间的得失。例如, 在识别诈骗案例的时候, 一般都是希望稍有嫌疑就拉入审核的名单, 因此召回率更重要, 较低的精度可以通过人工审核来弥补。而在知识问答系统中, 不见得需要返回很多条的候选, 只需要保证排在最前面的答案足够精准即可。
- 在这个关于美食的检索中, 系统 B 表现得比系统 A 更好, 因为从图 7-3 中可以看出无论为何种召回率, B 的精度都要高于或等于 A 的精度。这种图示化适合不同系统检索质量的对比。

精确率和召回率的概念很简单, 计算也方便, 因此广泛运用于信心检索的评估中。在此基础上, 人们又延伸和定义了其他几个常见的衡量指标, 即 F 值、前  $n$  精度、R 精度、平均精度均值、归一化折损累积增益、斯皮尔曼系数等。下面分别来看看。

### (1) F 值 (F-Measure)

F 值又称为调和平均, 它提供了一种将精度和召回率综合起来的数值方法, 可按照如下公式计算:

$$F(j, \beta) = \frac{(\beta^2 + 1) \times p_j \times r_j}{\beta^2 p_j + r_j}$$

其中,  $F(j, \beta)$  是用户阅读到第  $j$  个排位时的 F 值,  $p_j$  和  $r_j$  分别是用户阅读到第  $j$  个排位时的精度和召回率,  $\beta$  是控制二者相对权重的参数。当  $\beta$  为 1 时, F 值就具象为 F1 值。相对于将精度和召回率简单的加和, F1 值偏向于要求二者都要比较高。例如, 无论是精度 0.1+ 召回率 0.9, 还是精度 0.5+ 召回率 0.5, 这两种情况下加和都是 1.0。但是用户可能不希望过低的精度, 而是希望相对均衡, 这时候 F1 值就能体现差异了, 精度 0.1、召回率 0.9 的 F1 值是 0.18, 而精度和召回率都是 0.5 的 F1 值却能达到 0.5。如果参数  $\beta$  大于 1, 那么更看重召回率, 如果  $\beta$  小于 1 大于 0, 那么更看重精度。

### (2) 前 $n$ 精度 (P@n)

该指标考察前  $n$  个返回结果的精度。对于许多重要应用特别是搜索而言, 通常用户在意的是在第一页甚至只是排名前几位的结果, 这时需要评测排名靠前的若干个结果。例如, 最典型的 P@5 和 P@10 就是分别计算前 5 个和前 10 个返回结果的精度。该指标的优点是不需要统计全部相关数据集合的数目。但是, 相关数据的总数仍然会对 P@n 产生很大的影响, 甚至导致其不太稳定。

### (3) R 精度 (R-precision)

这个指标的主要思想是在第 R 个位置计算精度, 它在一定程度上弥补了 P@n 的不稳定

性，不过需要事先知道相关数据集 Rel，以及该集合的大小 R。其中 Rel 不一定是完整的相关集合，可以先将不同系统在一组实验中返回的前  $k$  个结果组成缓冲池，然后基于缓冲池进行相关性判定，从而得到 Rel 集合的一个预估。

#### (4) 平均精度均值 (Mean Average Precision)

该指标对 P@n 和 R 精度进行了扩充，可以在每个召回率水平上提供单指标结果。在众多评价指标中，平均精度均值 MAP 被证明具有非常好的区别性和稳定性。对于单个信息需求  $j$  而言，其  $MAP_j$  计算公式如下：

$$MAP_j = \frac{1}{|R_j|} \sum_{k=1}^{|R_j|} p(R_{j,k})$$

其中， $R_j$  是和信息需求  $j$  相关的数据集， $|R_j|$  表示该集合大小。 $R_{j,k}$  表示该集合中第  $K$  个相关的数据，而  $p(R_{j,k})$  表示用户检阅到第  $K$  个相关数据时的精度。如果有一组需求，那么整体的 MAP 计算则为：

$$MAP = \frac{1}{N} \sum_{j=1}^N MAP_j$$

其中， $N$  为信息需求集合的大小。

#### (5) 折损累积增益 (Discounted Cumulated Gain) 和归一化折损累积增益

前面的指标都是用于对给定集合进行精度评测的，没有考虑到相关数据的具体排位。例如，一个大小为 10 的集合中有 3 个相关，无论这 3 个对象排在 10 个的最前面还是最后面，精度都是 30%。但是从用户的角度而言，当然是希望相关的 3 个排在最前面。基于这种假设，检阅到第  $k$  个结果时，DCG 计算如下：

$$DCG_k = \sum_{l=1}^k \frac{2^{r_{j,l}} - 1}{\log_2(1+l)}$$

其中， $r_{j,l}$  表示第  $j$  个信息需求和第  $l$  个返回结果的相关性得分，相关则为 1，不相关则为 0。这里  $\frac{1}{\log_2(1+l)}$  可以看作权重，排名越靠前的对象其  $l$  值越小，这个权重就越大，那么

这个对象的相关性对 DCG 得分影响也就越大。DCG 的问题在于，不同的检索系统给出的返回结果可能有多有少，导致相互之间的 DCG 值没有可比性。人们就引入了 NDCG 的概念，公式如下：

$$NDCG_k = \frac{DCG_k}{IDCG}$$

分母 IDCG (Ideal DCG) 表示最理想情况下的 DCG 分值，就是检索结果完全按照黄金标准排序后，最大的 DCG 得分。这样，可以看出不同排序系统的优化空间，也可以对它们

的表现进行对比。

此外,从 DCG 和 NDCG 的公式可以看出,每个相关性不一定只有相关和不相关两个极端的打分,还可以取一个 0 ~ 1 之间的值,用于表示相关的程度。也正因为如此,这两个指标往往会应用在基于机器学习的排序方法中。

#### (6) 斯皮尔曼 (Spearman) 相关系数

第 6 章中已介绍过这个系数,它是依据两列成对等级的各对等级数之差来进行计算的,用于分析两个序列对于其维度排序位置认可的一致程度。在用于排序评测时,可将系统返回的排序和黄金标准做对比。和 NDCG 类似,斯皮尔曼系数也可以衡量相关的数据对象的排列是否更靠前,它能提供比基础精度和召回率更多的信息。

“这些评测的指标是挺精彩,可是我们还是需要黄金标准这样的答案才能实施吧?”

“没错,建立标准的答案集确实是很花费精力的。好在不同的领域中都有一些可以共享的数据,下面来讲一下文本检索领域中知名的测试集,其中会列出一些常见的标准测试集及相关的评测会议。”

□ Cranfield 测试集:这个集合来源于 20 世纪 50 年代末期进行的 Cranfield 实验,包括了 1398 篇空气动力学的期刊摘要,225 个查询及对应的相关性判断结果,是针对信息检索系统进行定量评价的早期测试集之一。但是对于现在的检索系统而言,这个数据规模实在太小,只能做一些尝试性的实验。

□ TREC 参考集:该测试集来源于全球著名的文本检索会议 (Text Retrieval Conference)。这个会议是由美国国家标准技术研究所组织的大型信息检索系统年度评测大会,始于 1992 年,目的就是为了解决当时信息检索研究缺乏公认标准的问题。其中,使用最为广泛的是 1992 年到 1999 年期间,8 次 TREC 即时 (Adhoc) 任务的测试集,总共包含了 189 万篇文章,450 个查询请求及其相关性判断。到了 2000 年,为了构建一个最大程度模仿 Web 环境的文档集合,在 TREC-9 中引入了 TREC Web 集合。值得注意的是,由于 TREC 的文档集往往都很大,文章数量众多,所以相关性的判定并不完整,一般都是根据系统返回的前  $n$  篇文章做判断。

□ NTCIR 参考集:NTCIR (NII Test Collection for IR Systems) 来自日本国立情报研究所的项目和研讨会,收集了由日文和英文专利组成的多种参考集,可用于专利检索、专利翻译和跨语言检索。

□ CLEF 参考集:测试集来自跨语言信息检索和评估研讨会 (Cross Language IR and Evaluation Forum),这里跨语言检索是指待检索的文档包含和查询不一样的其他国家语言。该集合支持多语言文档检索、多语言问答、交互式跨语言检索、跨语言图像检索等。

## 2. 分类的评估

还记得第 6 章介绍的数据挖掘和机器学习算法吗?与信息检索类似,没有哪一个算法



能够毫无误差地工作，因此在实际应用之前对其效果进行评估显得尤为重要。其中的监督式学习也是非常适合离线评估的，下面先来看一下如何评测分类的问题。值得一提的是，分类有两大类型：二分类和多分类。二分类是指判断数据对象属于或不属于一个给定的分类，而多分类则是指将数据对象判定为多个分类中的一个。多分类的评估策略会更复杂一些，由于我们可以将其转化为多个二分类问题来对待，所以先从二分类的评估入手，不过，在此之前要先认识一下表 7-2 中的混淆矩阵（Confusion Matrix）这个核心概念。

表 7-2 混淆矩阵示意图

实际的类	预测的类			
		Yes	No	合计
	Yes	True Positive (TP)	False Negative (FN)	Positive
	No	False Positive (FP)	True Negative (TN)	Negative
	合计	Positive'	Negative'	

现在来逐个解释一下这个矩阵中的元素，假设我们有一组标注好的数据集  $d$ ，并将其认定为标准答案。其中属于 A 类的数据称为正例（Positive），另外不属于 A 类的一部分数据称为负例（Negative），A 是正例和负例的并集，而且正例和负例没有交集。这时，我们通过一个分类算法  $c$  对这些数据进行判定，要判断一组数据对象是否属于 A 类。那么  $c$  判断属于 A 类的称为预测正例（Positive'），而不属于 A 类的称为预测负例（Negative'）。如果  $d$  标注为正例， $c$  也预测为正例，那么就叫作真正例 True Positive（TP）。如果  $d$  标注为正例， $c$  预测为负例，那么就叫作假负例 False Positive（FP）。如果  $d$  标注为负例， $c$  也预测为负例，那么就叫作真负例 True Negative（TN）。如果  $d$  标注为负例， $c$  预测为正例，那么就叫作假负例 False Negative（FN）。

细心的读者不难发现，两类问题的混淆矩阵和图 7-1 的集合图有着紧密的关系。没错，其实信息检索的相关性判定，可以认为是将检索结果分为“相关”或“不相关”的二分类问题。从下面的具体评测指标也可以看出分类和检索在离线评估上非常接近，这些指标依次是精度（Precision） $p$ 、召回率（Recall） $r$ 、准确率（Accuracy） $a$  和错误率（Error Rate） $e$ 。

$$p = \frac{TP}{TP + FP} = \frac{TP}{P'}$$
$$p \in [0, 1]$$

$$r = \frac{TP}{TP + FN} = \frac{TP}{P}$$
$$r \in [0, 1]$$

$$a = \frac{TP + TN}{P + N}$$
$$a \in [0, 1]$$

$$e = \frac{FP + FN}{P + N}$$
$$e \in [0, 1]$$

其中,精度和召回率与信息检索领域是一致的,因此F值也是适用的。但是,分类还会关注负例的预测情况,因此还定义了衡量整体预测的准确率和错误率,两者的和为100%。准确率和精度从中英文的角度上讲都是非常接近的,我们需要非常注意概念定义上的细微差别。如果有了这些,你还是觉得不太好理解,那就让我们再看看贯穿第6章的水果案例,这是一个典型的多分类问题,需要将水果分为至少三类:苹果、甜橙和西瓜。我们将其转化为3个二分类问题。表7-3再次列出10颗水果及其分类。

表 7-3 将 10 颗水果的特征值转为数字表示

特征 水果	形状 不规则圆: 1 圆形: 2 椭圆形: 3	外皮颜色 红色: 1 橙色: 2 绿色: 3	外皮纹理 无: 1 条纹: 2	重量 小于 200g: 1 200g 和 500g 间: 2 大于 500g: 3	握感 较硬: 1 较软: 2	口感 酸甜: 1 甜: 2
苹果 a	1	1	1	2	1	1
苹果 b	1	1	1	1	1	1
苹果 c	2	3	1	1	2	1
甜橙 a	2	2	1	1	2	2
甜橙 b	2	2	1	2	2	2
甜橙 c	1	2	1	2	1	1
西瓜 a	3	3	2	3	1	2
西瓜 b	3	3	2	3	1	1
西瓜 c	3	3	2	3	1	2
西瓜 d	1	3	2	3	2	2

假设我们有一个不是非常精准的分类算法  $c$  将这些水果按表 7-4 进行了分类。

表 7-4 分类算法  $c$  将 10 颗水果分为三类

分类里数据	预测的类		
	苹果	甜橙	西瓜
	苹果 a, 苹果 b, 甜橙 b, 西瓜 d	甜橙 a, 甜橙 c, 西瓜 c	苹果 c, 西瓜 a, 西瓜 b

下面先将其转化为 3 个二分类,依次为表 7-5、表 7-6 和表 7-7。

表 7-5 分类算法  $c$  对苹果的二分类

分类里数据	预测的类	
	苹果	非苹果
	苹果 a, 苹果 b, 甜橙 b, 西瓜 d	甜橙 a, 甜橙 c, 西瓜 c 苹果 c, 西瓜 a, 西瓜 b

表 7-6 分类算法 c 对甜橙的二分类

	预测的类	
	甜橙	非甜橙
分类里数据	甜橙 a, 甜橙 c, 西瓜 c	苹果 a, 苹果 b, 甜橙 b, 西瓜 d 苹果 c, 苹果 a 西瓜 b

表 7-7 分类算法 c 对西瓜的二分类

	预测的类	
	西瓜	非西瓜
分类里数据	苹果 c, 西瓜 a, 西瓜 b	苹果 a, 苹果 b, 甜橙 b, 西瓜 d 甜橙 a, 甜橙 c, 西瓜 c

逐个分析后，表 7-8 列出了表 7-5 的混淆矩阵。

表 7-8 苹果二分类的混淆矩阵

实际的类	预测的类		
	苹果	非苹果	合计
	苹果	苹果 a, 苹果 b	苹果 c
	非苹果	甜橙 b, 西瓜 c	甜橙 a, 甜橙 c, 西瓜 c, 西瓜 a, 西瓜 b
	合计	Positive	Negative

针对表 7-8，计算各项指标，如下：

精度  $p = \frac{2}{4} = 50\%$

召回率  $r = \frac{2}{3} = 66.67\%$

准确率  $a = \frac{7}{10} = 70\%$

错误率  $e = \frac{3}{10} = 30\%$

以此类推，对于表 7-6 中的甜橙分类问题，各项指标计算如下：

精度  $p = \frac{2}{3} = 66.67\%$

召回率  $r = \frac{2}{3} = 66.67\%$



准确率  $a = \frac{8}{10} = 80\%$

错误率  $e = \frac{8}{10} = 20\%$

对于表 7-7 的西瓜分类问题，各项指标计算如下：

精度  $p = \frac{2}{3} = 66.67\%$

召回率  $r = \frac{2}{4} = 50\%$

准确率  $a = \frac{7}{10} = 70\%$

错误率  $e = \frac{3}{10} = 30\%$

为了得到分类算法对整体分类的效果，可以将这些指标进行平均处理，宏平均（Macro Average）的计算如下：

精度  $p = \frac{\left(\frac{2}{4} + \frac{2}{3} + \frac{2}{3}\right)}{3} = 61.11\%$

召回率  $r = \frac{\left(\frac{2}{3} + \frac{2}{3} + \frac{2}{4}\right)}{3} = 61.11\%$

准确率  $a = \frac{\left(\frac{7}{10} + \frac{8}{10} + \frac{7}{10}\right)}{3} = 73.33\%$

错误率  $e = \frac{\left(\frac{3}{10} + \frac{2}{10} + \frac{3}{10}\right)}{3} = 26.67\%$

如果是微平均（Micro Average）处理，那么需要将 3 个混淆矩阵数中的 TP、FN、FP 和 TN 值相加，见表 7-9。

表 7-9 合并后的混淆矩阵

		预测的类	
	Yes	No	合计
Yes	6	4	Positive
No	4	16	Negative
合计	Positive '	Negative '	

如此，整个数据集分类的各个指标计算如下：

$$\text{精度 } p = \frac{6}{10} = 60\%$$

$$\text{召回率 } r = \frac{6}{10} = 60\%$$

$$\text{准确率 } a = \frac{22}{30} = 73.33\%$$

$$\text{错误率 } e = \frac{8}{30} = 26.67\%$$

其中，对于准确率和错误率，宏观和微观的均值都一样。

综上所述，我们可以看到分类和信息检索一样，比较适合离线的测评。类似地，从事相关研究的人员也在不断地建立和贡献测试的标准集合。下面列出了一些标准集合以供参考。

□ 20NG 新闻组 (20 NewsGroups)：如果是小范围的测试，这个集合也许是一个不错的选择，包括大约 20 000 条发布在 Usenet 新闻组上的消息。Usenet 平均分为 20 个不同的新闻组，每个新闻组提供约 1 000 条消息。

□ Reuters-21578：在分类实验中使用非常广泛的一个文档集，顾名思义，是由路透社在 1987 年的发表 21 578 篇新闻稿组成的，包含多个和经济有关的分类。通常采用 ModApte 的划分标准，将 9 600 多篇文档作为训练，3 200 多篇文档作为测试。

□ RCV (Reuters Corpus Volumes)：RCV 版本 1 是一个由超过 806 000 篇人工分类的新闻报道组成的数据集，包含 103 个主题分类。因为这个数据集规模相比 Reuters-21578 要大得多，所以创建者们希望用此数据集逐步替换 Reuters-21578。其版本 2 对一部分错误数据进行了修正，因此更加可靠一些，其规模和丰富的标注也为相关科研提供了更好的基础。

□ OHSUMED：这个是由美国国家医药图书馆维护的医疗文献组成的，是 MEDLINE 数据库的一个子集。它拥有 348 566 篇医学文献，覆盖了 270 种学术期刊。除了文档，OHSUMED 还包含了医生在真实病例中提出的 100 多个查询请求。

顺便提一下线性回归，如果回归用于预测离散的分类标签，那么评估的方式就如前所述。如果是用于预测连续值，那么一般采用预测值和真实值之间的误差来进行评估。

### 3. 监督式学习的交叉验证 (Cross Validation)

对于基于监督式的学习，包括分类模型、基于回归的排序模型等，除了定义评估的指标之外，还需要考虑一个很实际的问题：我们该如何选择训练数据集和测试数据集？离线测试的时候，我们需要预留一部分的标注样本作为测试。然而，黄金标准虽然为人们准备好了标好的数据，却不一定告诉人们如何选择训练和测试的划分。这种划分对最终评测的结论可能会产生很大的影响，主要原因有两个：

□ 训练样本的数量决定了模型的效果。如果不考虑过拟合的情况，那么对于同一个模

型而言,一般来说训练数据越多精度越高。例如,方案A选择90%的数据作为训练样本来训练模型,剩下10%的数据作为测试样本,而方案B正好颠倒,只用10%的数据作为训练样本,测试剩下的90%的数据。那方案A测试下的模型准确率很可能比方案B测出的模型准确率要好很多。模型是一样的,但训练和测试的数据比例导致了结论的偏差。

- 不同的样本有不同的数据分布。假设方案A和B都取90%作为训练样本,但是A取的是前90%的部分,而B取的是后90%的部分,二者数据分布不同,对于模型的训练效果也可能不同。同理,这时剩下的10%的测试数据分布也不相同,这些都会导致评测结果不一致。

鉴于此,人们发明了一种称为交叉验证(Cross Validation)的划分和测试方式。其核心思想是在每一轮中,都拿出大部分数据实例进行建模,然后用建立的模型对留下的小部分实例进行预测,最终对本次预测结果进行评估。这个过程反复进行若干轮,直到所有的标注样本都被预测了一次而且仅一次。用交叉验证的目的是为了得到可靠稳定的模型,最常见的形势是留一验证和K折交叉。留一验证(Leave One Out)是交叉验证的特殊形式,意指只使用标注数据中的一个数据实例来当作验证资料,而剩余的则全部当作训练数据。这个步骤一直持续到每个实例都被当作一次验证资料。而K折交叉验证(K-fold Cross Validation)是指训练集被随机的划分为K等分,每次都是采用(K-1)份样本用来训练,最后1份被保留作为验证模型的测试数据。如此交叉验证重复K次,每个1/K子样本验证一次,通过平均K次的结果可以得到整体的评估值。假设有数据集D被切分为K份( $d_1, d_2, \dots, d_k$ ),交叉过程则按如下形式来表示:

$$Validation_1 = d_1 \quad Test_1 = d_2 \cup d_3 \cup \dots \cup d_k$$

$$Validation_2 = d_2 \quad Test_2 = d_1 \cup d_3 \cup \dots \cup d_k$$

...

$$Validation_k = d_k \quad Test_k = d_1 \cup d_2 \cup \dots \cup d_{k-1}$$

K的值一般取5到30,而10最为常见。随着K值的增大,训练的成本就会变高,但是模型可能会更加精准。当标注集的数据规模很大时,K值可以适当小一些,反之则建议K值适当地取得大一些。

#### 4. 聚类的评估

在第6章中,还介绍了非监督式学习最经典的算法:聚类。其目标是将相似度高的数据对象聚集到同一个群组,而不够相似的分隔在不同的群组。不过,在实际应用中这些相似度标准导致的结果质量是否足够高呢?是否就一定符合用户的预期呢?最为直接的衡量方法是让用户试用并给出反馈,但是这需要在访谈上耗费大量的时间和人力。与此同时,聚类的离线评估又缺乏黄金标准这样的答案集合。

“对啊,这样听起来好像聚类没有办法做离线的评估哦。”

“也不尽然，虽然很有挑战，但是我们还是可以尝试一些迂回的方法，这里介绍一个最为常用的外部准则（External Criterion）法。”

其实所谓的外部准则法，就是借鉴分类问题中的黄金标准和评价指标，计算聚类结果和已有标准分类的吻合程度。其基本假设是：希望每个聚出来的群组，其组员尽量来自一个分类，尽量“纯净”。举个例子，我们对水果案例中的 10 颗水果进行聚类，2 个聚类算法在结束后分别得到下面的分组。

#### 算法 A

{1, 8, 10}, {4, 7}, {2, 3, 5, 6, 9}

#### 算法 B

{1, 8}, {10}, {4, 7}, {2, 5, 6}, {3, 9}

评估之前是无法知道它们的标签的，需要评估的时候，拿出分类的标签作为参考答案，我们可以得到：

#### 算法 A

{ 苹果 a, 西瓜 b, 西瓜 d }, { 甜橙 a, 西瓜 a }, { 苹果 b, 苹果 c, 甜橙 b, 甜橙 c, 西瓜 c }

#### 算法 B

{ 苹果 a, 西瓜 b }, { 西瓜 d }, { 甜橙 a, 西瓜 a }, { 苹果 b, 甜橙 b, 甜橙 c }, { 苹果 c, 西瓜 c }

这样就能衡量每个群组的纯度。在此之前，首先简短回顾下熵（entropy）的概念，其将在第 6 章决策树分类中有所提及。它是用来刻画给定集合的纯度的，如果一个集合里的元素全部是同一个分类，那么熵就为 0，表示最纯净。如果元素分布在不同的分类里，那么熵就是大于 0 的值，而且随着分类的增多，元素的分布就越均匀，熵值也就越大，表示混乱程度越高。其计算公式如下：

$$Entropy(P) = -\sum_{i=1}^n p_i \times \log_2 p_i$$

其中  $n$  表示集合中分类的数量， $p_i$  表示属于第  $i$  个分组的元素在集合中的占比。有了用于分类的训练数据，以及熵的定义，就可以计算每个聚类的纯度了。对于群组 { 苹果 b, 苹果 c, 甜橙 b, 甜橙 c, 西瓜 c } 而言，共 5 个对象，苹果有 2 个占 0.4，甜橙有 2 个也占 0.4，西瓜占剩余的 0.2，其熵值约是 1.52：

$$Entropy(P) = -(0.4 \times \log_2 0.4 + 0.4 \times \log_2 0.4 + 0.2 \times \log_2 0.2) \approx 1.52$$

由于聚类结果有多个群组，最后进行加和平均：

$$Entropy(P) = \frac{1}{N} \sum_{i=1}^n Entropy(P_i)$$

那么依据算法 A 聚类的结果最终整体熵值为：

$$Entropy(P) = \frac{(0.92 + 1 + 1.52)}{3} \approx 1.15$$



依据算法 B 聚类的结果最终整体熵值为：

$$Entropy(P) = \frac{(1+0+1+0.92+1)}{5} \approx 0.78$$

不过，由于聚类并不会像分类那样指定类的个数，因此这种最基础的熵值评估存在一个明显的问题：它会偏向于聚出更多的群组，这样评测出的结论是算法 B 会优于算法 A。但果真如此吗？西瓜 b 和 d 被算法 A 聚集了，但被算法 B 给拆分了。最极端的情况就是每个数据对象就是一个群组，这样全体的熵为 0。但是这并没有实际的意义，因为没有产生任何的聚类效果。所以可将整体熵的计算公式修正为如下形式。

$$Entropy(P) = Entropy(C) \times \frac{1}{N} \sum_{i=1}^n Entropy(P_i)$$

这里假设聚类的划分是合理的， $Entropy(C)$  是基于这个划分计算的熵值，如果一个算法聚出来很多细小的群组，那么  $Entropy(C)$  一定会很大，这就好比是进行了一个惩罚。

这样一来，算法 A 的  $Entropy(C)$  计算就会变成如下形式：

$$Entropy(C) = -(0.3 \times \log_2 0.3 + 0.2 \times \log_2 0.2 + 0.5 \times \log_2 0.5) \approx 1.49$$

$$Entropy(P) = 1.15 \times 1.49 = 1.71$$

算法 B 的  $Entropy(C)$  计算则变成如下形式：

$$Entropy(C) = -(0.2 \times \log_2 0.2 + 0.1 \times \log_2 0.1 + 0.2 \times \log_2 0.2 + 0.3 \times \log_2 0.3 + 0.2 \times \log_2 0.2) \\ \approx 2.25$$

$$Entropy(P) = 0.78 \times 2.25 = 1.76$$

“这样看来，熵不仅可以用于决策树的模型训练，还可以用于聚类的效果评估？”

“没错，在算法领域中，很多指标或系数，不仅可以用于建模，还可以用于评测，是多功能的。除了熵，我们还发现了斯皮尔曼这样的系数，不仅可以用于聚类的相似度计算，还可用于对信息检索中的排序进行评估，不是吗？大宝，你再想想看，我们所介绍的还有哪些指标也是如此呢？”

除了分类标注，还可以借鉴分类的评价指标，例如准确率（Accuracy）和 F 值。不过前提是需要将聚出的群组和某个标注的分类进行对应，最基本的方法是看组员的大多数属于哪个分类，然后以这个分类作为答案，群组作为“分类的预测”。这样问题就转化成为分类的离线评估了，具体的前面已经有所阐述，这里不再重复。

## 7.1.2 非离线的评估

我们要意识到，实际生产环境拥有其特定的应用场景，某些因素已经远远超出了离线评估的范围，因此离线评估得出的精度、召回率、准确率、错误率和其他相关的指标，往往无法满足人们的需求。推荐系统就是很好的例子。推荐由信息检索、数据挖掘等综合发展而

来，逐步形成了自己的一套理论体系，尤其是评价的指标。和普通的检索系统相比，理想的推荐系统更能充分地挖掘用户的潜在需求，发现长尾（Long Tail）物品，覆盖尽量多的类别，以避免过多的重复和热门品项，因此除了精度、召回率、准确率和错误率，推荐系统还需要从如下几个方面给出评估参考。

- 多样性：推荐结果能覆盖用户不同的兴趣领域。可以通过推荐列表中的两两相似性来衡量，物品之间的相似度越低，那么多样性就越高。
- 新颖性：如果用户对于推荐的物品了解甚少，那么可以认为该物品足够新颖，一般都是长尾物品。可以通过计算推荐结果的平均热门程度来预估。
- 惊喜度：如果推荐的物品和用户的历史兴趣关联不大，而用户又非常满意，那么可以认为该推荐给用户带来了惊喜。

综上所述可以看出，多样性尚可依据离线测试集合来评判，但是新颖度和惊喜度都是和用户紧密相关的。而每位用户的历史行为、兴趣爱好、评价方式都可能不同，而且会随着时间的推移而不断变化，这就使得构建离线的评判几乎变成了不可能的任务。

此外，电子商务系统中的商品排序，或者是在线竞价广告也面临着同样的问题。考虑到其涉及的用户偏好、季节地域、竞拍价格等因素，因此无法构建离线的评测集合，即使能构建，成本也会非常巨大。因此，实际不太可能对这种系统进行离线环境的评估，而只能依靠非离线的方式来完成。而非离线的方式主要又分为用户访谈和在线评估（直接测试线上的系统）。

### 1. 用户访谈

用户访谈也称为用户调研（User Study），通常是招聘一组测试对象，并要求他们执行一些需要与系统进行交互的任务。在被测对象执行任务的时候，工作人员观察并记录他们的行为，例如，完成了任务的哪些部分，结果的准确性如何，或者是执行任务所花费的时间。在任务完成之前、期间和之后的场景下，工作人员还可以提出很多定性的问题。这些问题用于收集那些不能被直接观测到的数据，例如，被测对象是否喜欢系统的用户界面，或者用户觉得完成任务是否足够轻松等。此外，还有些客观的数据也会被自动地收集，比如用户的浏览和点击行为，甚至是采用眼球追踪系统观测到的用户阅读行为等。和离线评测不同，用户访谈允许测试用户和系统之间进行交互，而这可以获取到更多的信息，是其明显的优势。当然，这种方式也存在如下几个问题：

- 访谈的执行需要很高的代价。招募大量的受测人员，要求他们进行大量的任务，并给出反馈，会产生相应的时间成本和财务成本。因此访谈往往受限于极少数人的小组内。
- 对人选的限制又带来另外一个问题，被测试的人员是否具有足够的代表性？只有当参与者和真实的用户尽量相似时，才有可能获得更可靠的评估结论。
- 即使被测对象代表了真实用户，结果可能还是有偏差，因为他们被明确告知参加一个实验，无论是志愿者还是付费的参与者，更倾向于迎合实验发起人的口味，潜意

识里会提供有利于假设的证据。因此,在采集数据前要尽量避免实验目的的泄露。

考虑到这些因素,一些大公司的评测开始采用众包(Crowdsourcing)的形式来完成。众包是连线杂志2006年发明的一个专业术语,用于描述互联网带来的一种新生产组织形式。即企业利用互联网来将工作分配出去,发现创意或解决技术问题。参与的志愿员工要具备完成任务的技能,愿意利用业余时间来参与工作,满足于对其服务收取小额报酬,或者暂时并无报酬,但未来可能获得更多报酬的前景。这些特点在一定程度上缓解了用户访谈所面临的问题。

最后,调查问卷是访谈中最常用的工具。测试对象在执行任务之前、期间和之后,我们都需要问一些关于用户体验的问题。这些问题通常是针对很难量化的指标进行的,例如他们心里的想法、心情和感受等。编写问卷绝对是个艺术,可能还需要一些心理学的知识。

## 2. 在线评估

快速了解用户访谈的方式后,下面来阐述评测的最后一种方式,可能也是大数据时代最有价值的一种方式:在线评估。在很多实际应用中,系统设计者希望能影响最终的用户行为,并对业务产生积极的作用,他们更关注评测用户和多个不同系统的直接交互。因而真实的用户在真实的系统上执行真实的任务,将会提供更强有力的证据,这也是在线评估的由来。

“小明哥,那这个在线评估具体的好处在哪里?离线评估或用户访谈应该也能完成这些吧?”

“首先说说相对于离线评估而言它的好处。在线评估和用户访谈一样,都是记录真实用户的行为,适合规模大、变化快的数据,为那些很难建立离线标准的实验提供了可能,这是其优势之一。此外,和用户访谈相比,在线测试也有自己的优劣势。用户访谈中获得的反馈更为详细和明确,包括使用者的想法和建议,互动的方式还允许研究人员进一步提问并收集答案。而在线测试一旦开始运行,基本上都是全自动化的过程。被测试的用户甚至都不知道他/她已经参与其中了,自然也不会主动提供定性的答案。研究人员只能事前做好实验的假设和设计,事后再根据系统自动记录的数据进行分析,这是其相对用户访谈而言的劣势。举个例子,某购物网站想测试哪种商品排序会产生更高的销售额。我们只能假设在不同的方案下,最终的销量差异会体现方案的优劣,但实际中影响销量的因素很多,不见得就是排序算法本身所导致的(后面也会具体介绍)。当然,在线测试比用户访谈更强大的地方在于规模。试想一下,有位领导要求你在1个月内,测试100种方案,访谈50万用户,你一定会觉得他/她疯了。而对在线测试而言,无论是要测试多长的时间、多少种方案、还是测试多少个用户,它都无需花费额外的成本在被测试者的身上。你所做的一切只是每天换一下方案的配置文件,以及随后的数据分析。是不是很棒?!目前很多大型的互联网公司都是采用在线测试作为主要的评估手段。”

“还有个问题,这在线测试虽然好,可是如果线上的表现有所提升或下降,我们怎么知道这个改变一定是新方案导致的呢?”



“很棒的问题！提到点子上了，我来解释下。”

的确，在线测试看上去很美，但是还有一个最大的挑战是需要克服的，那就是如何排除非测试因素的干扰。为了便于理解，我们先看下日常生活中最生动的例子：股票市场。很多人都有炒股的经历，但是要赚钱并非易事，最主要的原因就是影响股价的因素太多了，包括国家政策、国际局势、行业变化、公司业绩等。最后的结果就是，股价的精准预测基本上是不可能的，尤其是短期内的波动。当然，我们这里不是教大家如何炒股，而是让大家明白：在线测试面临同样的问题。看看如图 7-4 所示的示意图。

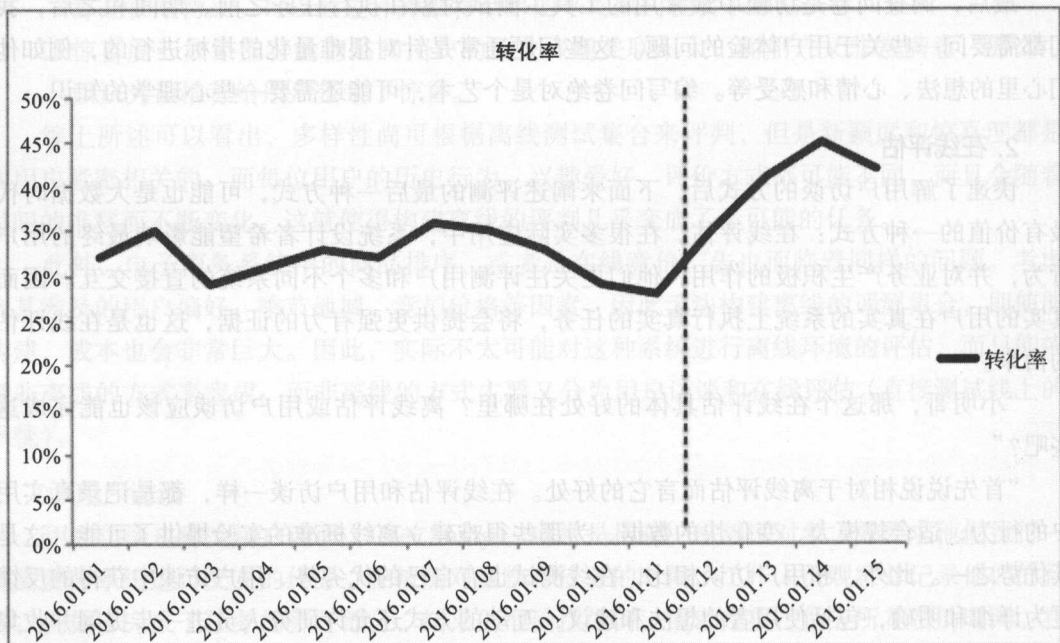


图 7-4 某转化率趋势变化图

从图 7-4 中可以看出从 2016 年 1 月 12 日开始，某个转化率曲线的趋势发生了明显的变化，而这天恰好上线了一个新版的技术方案 A，那么转化率上涨一定是新方案导致的吗？其实不然，很有可能是 1 月 12 日开始有个大型的促销活动使得价格有大幅下降，或者是有个和大型企业的合作引入了很多优质顾客等。如果将时光倒流，我们取消 12 日上线的技术方案 A，然后用虚线表示在这种情况下的转化率曲线，就可以得到图 7-5 中的对比。

从图 7-5 可以发现，不用方案 A，反而能获得更好的转化率表现。所以简单地使用在线测试的结果往往会导致错误的结论，我们需要一个更健壮的测试方法。这里必须要提到 AB 在线测试这个不错的选择。

AB 在线测试，简单来说，就是为同一个目标制定两个或多个方案，比如两种页面、两种流程、两个算法等。多种方案同时上线，让一部分用户使用 A 方案，另一部分用户使用



B 方案，记录下用户的使用情况，看哪个方案更符合预期。其目的在于通过科学的实验设计、代表性的采样样本、分割的小流量测试等方式来获得具有统计意义的实验结论，并确信该结论在推广到全部流量时是可信的。图 7-6 展示了整个流程，在进行 AB 测试的时候，用户请求在服务器端做了区分，一部分流量得到结果 A，另一部分得到结果 B。与此同时，相应的用户反馈数据也会记录到日志里。

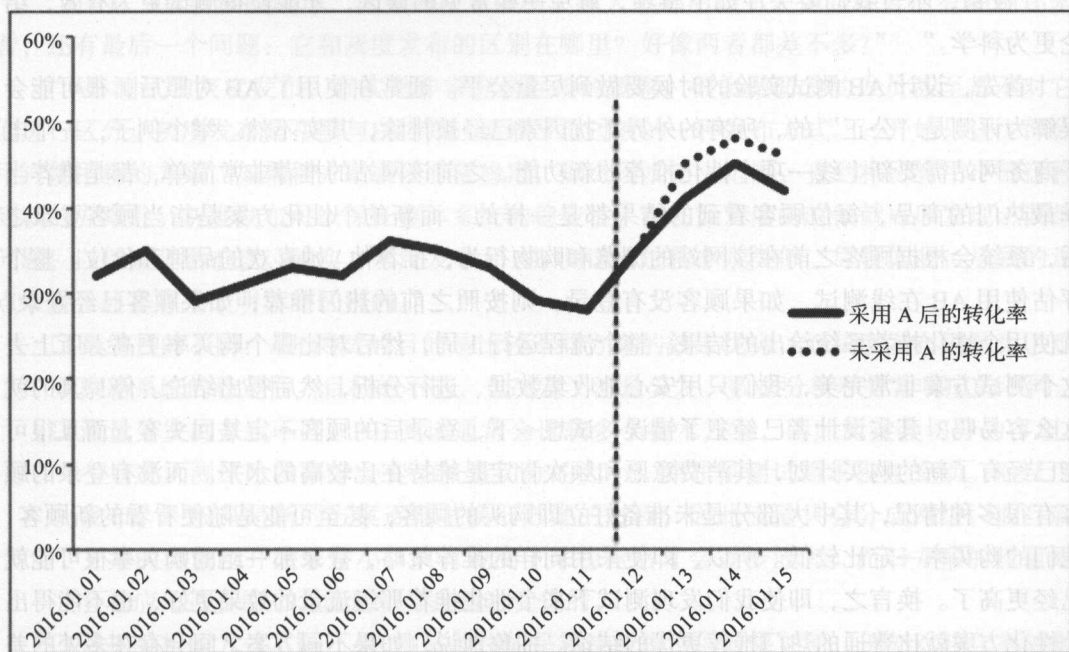


图 7-5 转化率趋势对比图

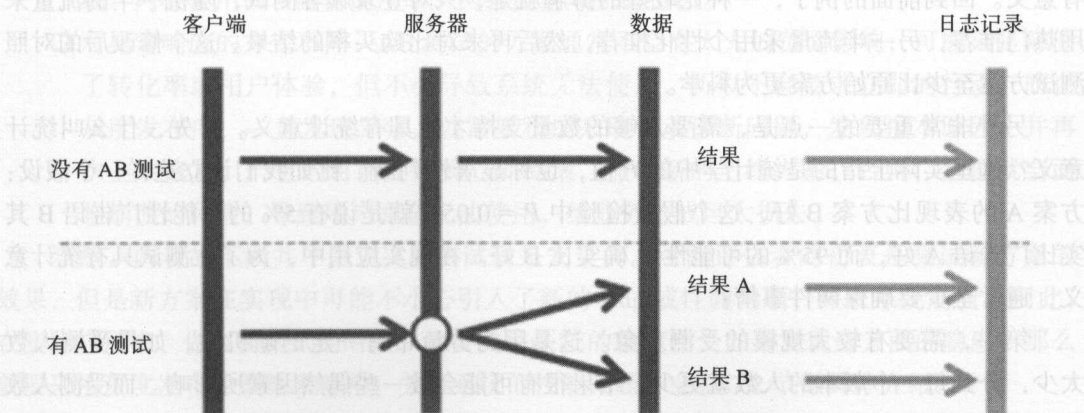


图 7-6 AB 在线测试流量切分示意

从这个流程和架构中，我们很容易理解 AB 在线测试相对于普通在线测试的好处在于：

- 在流量足够的前提下，可以同时测试多个方案，大幅提升测试的效率。
- 在很大程度上排除外来因素的干扰，能提升结论的可靠程度。
- 可以控制受影响的流量规模，降低不良方案可能产生的负面影响。

“哇，看来 AB 在线测试是大型测试的必备神器啊！”

“没错，不过我们要关注如下事项，避免一些常见的误区，才能使得测试更为有效，结论更为科学。”

首先，设计 AB 测试实验的时候要做到尽量公平。通常在使用了 AB 对照后，很可能会误解为评测是“公正”的，所有的外界干扰因素已经被排除，其实不然。举个例子，一个电子商务网站需要新上线一项个性化推荐的新功能。之前该网站的推荐非常简单，都是推荐当季最热门的商品，每位顾客看到的结果都是一样的。而新的个性化方案是指当顾客登录之后，系统会根据顾客之前在该网站的浏览和购物行为，推荐他\她喜欢的品牌和价位。整个评估使用 AB 在线测试，如果顾客没有登录，则按照之前的热门推荐；如果顾客已经登录，就使用个性化推荐系统给出的结果。整个流程运行 1 周，然后对比哪个购买率更高。听上去这个测试方案非常完美，我们只用安心地收集数据、进行分析，然后得出结论。停！真的就这么容易吗？其实设计者已经犯了错误，试想一下，登录后的顾客一定是回头客，而且很可能已经有了新的购买计划，其消费意愿和频次肯定是维持在比较高的水平。而没有登录的顾客有很多情况，其中大部分是未准备好立即购买为顾客，甚至可能是随便看看的新顾客，他们的购买率一定比较低。所以，即使采用同样的推荐策略，登录那一组的购买率很可能就已经更高了。换言之，即使我们发现测试下来个性化推荐那组流量的效果更好，也不能得出个性化方案就比普通的热门推荐更优的结论。抽象地说，如果不同方案之间会存在多处的差异，那么这样的情况一般不太适合直接做 AB 测试，因为可变量太多了，相互会有较多的干扰。最好的方法是将这些不同逐步拆解和细分，每次对单个变量进行测试，这样结论才会更有意义。回到前面的例子，一种比较好的拆解就是，只对登录顾客测试，随机一半的流量采用热门推荐，另一半流量采用个性化推荐，然后再来对比购买率的结果。这个修改后的对照测试方案至少比原始方案更为科学。

另外非常重要的一点是，需要足够的数量支持才能具有统计意义。首先，什么叫统计意义？这里实际上指的是统计学中的  $P$  值，也称显著性判断。比如我们设立这样一个假设：方案 A 的表现比方案 B 好，这个假设检验中  $P = 0.05$ ，就是说有 5% 的可能性广告语 B 其实比广告语 A 好，而 95% 的可能性 A 确实比 B 好。在现实应用中，为了让测试具有统计意义，通常至少要确保两件事情：

第一，需要有较大规模的受测对象。这是因为分流带有一定的随机性，如果受测人数太少，分到每一个版本的人数就更少，结果很有可能会被一些偶然因素所影响。而受测人数较多时，根据大数定理，得到的结果会更接近于真实数据。对于互联网而言一般就是足够多的流量，有较高的 UV (Unique Visitor) 和 PV (Page View)。

第二，确保足够的测试时间和频率。虽然我们在设计测试方案的时候已经注意尽量保持公平，但是完全排除现实中的干扰因素几乎是不可能的。因此，一两次的结果差异可能还是由偶然性导致的，不能得出肯定的结论。好在在线测试的强项就是规模，要实现这点并不困难。不得不承认，公平性和统计意义只能是在一定程度上做出保证，需要根据应用场景和部署经验来综合确定，没有一个放之四海而皆准的标准。

“好的，AB测试的原理、流程和注意事项，我大致都明白了。但是综合所有的描述来看，还有最后一个问题：它和灰度发布的区别在哪里？好像两者都差不多？”

“AB测试和灰度发布，确实是很容易混淆的概念，很多做研发的人员甚至都不对它们进行区分。灰度发布是在黑与白之间，能够平滑过渡的一种发布方式。一开始只让少部分用户使用新的方案。如果新方案表现非常稳定，那么逐步扩大范围，直到将所有用户都迁移到新的方案上。灰度发布可以保证整体系统的稳定性，在初始灰度的时候就可以发现、调整问题，以保证其影响度。确实，灰度发布和AB测试的过程非常类似。尽管如此，从我的观点来看，还是有几个明显的区别的。”

□ 最终目的不同：AB测试的目的是比较不同方案的效果优劣，而灰度发布则是为了观察新系统的稳定性。目标的不同，也决定了流量分发、数据分析和异常响应的不同。

□ 流量分发的方式不同：AB测试通常会比较多个技术方案，因此流量会切分为多组，一旦进入测试阶段很少会变动流量切分的大小，除非是测试需求发生变动。而灰度发布，一般只是新的方案逐渐替换旧方案的过程，因此流量只会分成两组，而且新方案的流量刚开始非常小，只有观测下来稳定后才会逐步放大，而旧方案的流量则会相应地逐步缩小，流量切分是随时变化的。

□ 数据分析的方式不同：AB测试关注的是算法或模型的效果如何，例如对转化率的影响，因此都是先收集数据，定期再进行事后的分析，并不关心系统的实时状态。而灰度发布需要验证系统的稳定性，因此实时的监控非常重要，包括计算机的CPU负载、内存使用率、磁盘使用率、网络流量等关键指标。

□ 对异常响应的方式不同：AB测试的方案通常不会有太大的负面影响，可能只是降低了转化率或用户体验，但不会导致系统无法使用，所以不会要求立即的行动。而在灰度发布中，一旦发现异常，就需要积极采取行动诊断问题，发现根本原因，并再次上线修复，严重的可能还需要立即回滚，撤销新的上线方案。否则，系统就会面临崩溃的风险，导致用户完全无法使用，给业务造成巨大损失。

当然，两者的不同并不是说AB测试就是安全的。可能评估者的初衷是测试新方案的效果，但是新方案在实现中可能不小心引入了新的Bug或性能问题，导致系统崩溃。因此，在实际应用中，做AB测试的时候，也要关注系统的稳定性，如果发现这方面的隐患，那么就要采用灰度发布的策略来对待。关于系统的性能问题，会在7.2节做具体介绍。

至此，我们理解了三种评估的主流方式：离线、用户访谈和在线。它们也不一定是相互对立的，通常我们需要灵活地将它们结合起来使用。比如说，先进行一定的离线评估和用户



访谈，保证新方案达到基本的效果或预期，然后再大规模地投放到线上进行实际的测试，这样就会大大降低不良方案可能带来的风险。

## 7.2 性能评估

如果说上节介绍的效果评估是针对数据处理质量的测量，那么本节即将介绍的性能评估则是针对数据处理速度和稳定性的测量。这两者是相辅相成，缺一不可的。举个平日中生动的例子，玩家们在玩电子游戏的时候，期望有绚丽的3D画面，这就是对效果的预期，同时，他们也希望畅快的游戏速度，这就是对性能的预期。同样，在大数据的处理中，一个分类模型不仅要能准确地预测新的数据属于哪个类别，也要在用户预期的时间内给出答案。

所以，这里所说的系统性能是指计算机系统的性能，其评估就是采用测量、模拟、分析等方法 and 工具，研究计算机系统的生产率、利用率、可用率、响应特性等，将看不见摸不着的性能转换为人们能够量化和评比的客观指标。它最早起源于20世纪60年代中期，原因是随着人们对多任务、多用户计算机系统的深入使用，大家发现这些系统表现出来的实际性能不一定能达到预计的目标，从而引发了对计算机系统性能评价的研究。由于计算机系统从设计、开发到最终的部署，要经过若干步骤，因此影响系统性能的因素也就很多了，依次主要分为算法理论上的计算复杂度，开发实现的方案，以及硬件设备的规格，如图7-7所示。

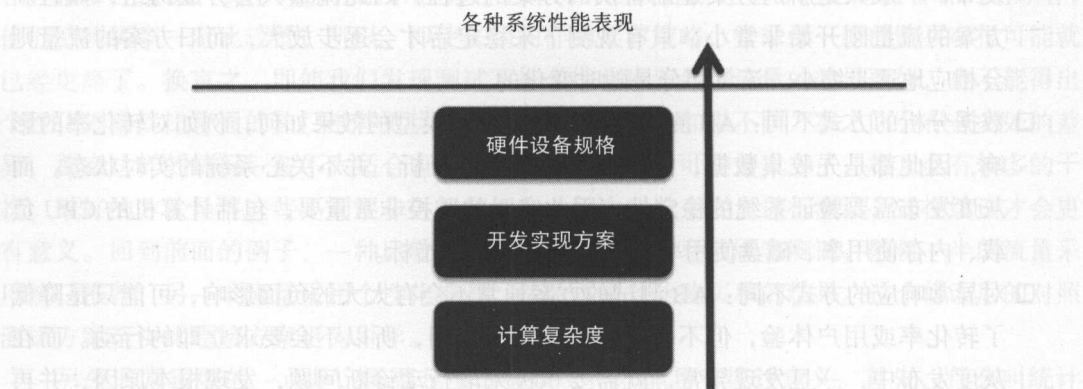


图 7-7 影响性能表现的主要因素

如果将整个系统的构建比作建造房屋，那么计算复杂度评估相当于在蓝图设计阶段，从理论上预估整个建造的工作量和工期，而开发实现就关乎水泥浇筑这样具体的施工工艺了，硬件设备就相当于打桩机和起重机这样的生产工具。优秀的研发人员能够选择合理的技术方案，大大降低计算复杂度和开发实现的难度，保证在同样硬件资源下的性能表现，是为上策。因此这里并不讨论和硬件相关的内容。对于计算复杂度，我们先阐述基本概念，以及如何从理论上进行评估。而对于应用开发，这里只是给出常见评测指标的概念。

## 7.2.1 计算复杂度

这里的计算复杂度会借用算法复杂度的概念，主要从系统的执行时间和所需要占用的存储空间两个方面来衡量，分别称为时间复杂度（Time Complexity）和空间复杂度（Space Complexity）。时间复杂度是一个函数，它定量地描述了系统的运行时间。这是一个关于代表算法输入值的字符串的长度的函数。时间复杂度常用大 O 符号来表述。使用这种方式时，时间复杂度可被称为是渐近的，它考察当输入值大小趋近于无穷时的情况。空间复杂度则是对一个系统在运行过程中临时占用存储空间大小的量度，同样可以用大 O 符号来表述。

为了更好地理解这两个概念，先简短回顾下第 5 章介绍的倒排索引构建和查询，并分析这些步骤中的时间和空间复杂度。倒排索引就像是给图书馆中的图书贴分类标签一样，维护着单词到包含其的文档之间的关系，并采用类散列表的结构来存储，输入后可以进行快速查找。因此，你会发现通过关键词查询起来就像在图书馆里根据书架上的标签找书一样方便快捷，效率得到大大提升。假设有表 7-10 中所示的 5 篇文章，我们可以得到类似表 7-11 的倒排索引<sup>①</sup>。对于多个关键词的查询，只需要取每个关键词对应的文档集合的交集即可。

表 7-10 5 篇文章样例

文章 ID	内容
1	最上瘾的绝味川菜
2	大厨必读系列：经典川菜
3	舌尖上的川菜
4	舌尖上的中国味 在家吃遍八大菜系
5	舌尖上的历史

表 7-11 5 篇文章分词之后所建立的倒排索引，包含位置信息

关键词 ID	关键词	文章 ID: 位置
1	最	1:1
2	上瘾	1:2
3	的	1:3, 3:3, 4:3, 5:3
4	绝味	1:4
5	川菜	1:5, 2:5, 3:4
6	大厨	2:1
7	必读	2:2
8	系列	2:3
9	经典	2:4
10	舌尖	3:1, 4:1, 5:1
11	上	3:2, 4:2, 5:2
12	中国	4:4

① 如果对具体过程有些遗忘，可以参考第 5 章的相关内容。

(续)

关键词 ID	关键词	文章 ID: 位置
13	味	4:5
14	在家	4:6
15	吃遍	4:7
16	八大	4:8
17	菜系	4:9
18	历史	5:4

为了分析计算的复杂度，我们将建立倒排索引的过程进行大致分解，对于  $n$  篇文档集合中的每一篇，进行如下步骤。

1) 全文扫描并进行中文分词。

2) 切分出若干个关键词，然后针对每个关键词在现有的倒排索引中查找该关键词，例如在表 7-11 中发现“舌尖”在第 10 行，并在查找到相应的位置后进行文档列表的插入，更新索引结果。例如在表 7-11 中的第 10 行，插入 5:1 的信息。

然后重复步骤 2，直至所有文档结束为止。

假设集合里的文章最多能切出  $l$  个单词数，次切词是一次操作，散列表查找是一次操作<sup>①</sup>，而更新文档列表的插入是一次操作，那么构建索引的整个过程的操作总数则为  $n \times (3 \times l)$ 。如果是在最坏的情况下，那么时间复杂度就是  $O(n \times l)$ ，3 作为常量被忽略。另外，假设最后的词典里有  $v$  个词条，最坏情况下，每个词条对应的文档列表是全集，大小为  $n$ ，那么需要  $v \times n$  个存储空间，空间复杂度就是  $O(v \times n)$ 。

对于在线的查询阶段，过程大致可分解为如下步骤：

1) 对于查询中的每一个关键词，查询倒排索引里对应的文档集合，例如在表 7-11 中取出第 10 行“3:1, 4:1, 5:1”。

2) 重复步骤 1 直到所有关键词结束。

3) 为了简化起见，假设关键词直接都是 AND 的布尔关系，那么取所有的关键词对应文档集合的交集。例如，对“川菜”的“1:5, 2:5, 3:4”和“舌尖”的“3:1, 4:1, 5:1”两者取交集。

假设查询最多包含  $k$  个关键词，那么查找倒排索引需要  $k$  次操作，对应的文档列表其集合数最大为  $n$ ，取交集的操作次数也为  $n$ ，那么操作总次数为  $k + n$ ，时间复杂度是  $O(k \times n)$ ，而应用于存储文档列表所需要的存储空间为  $n$  个，所以空间复杂度是  $O(n)$ 。

从离线索引和在线查询的复杂度对比可以看出：

□ 离线索引阶段的时间复杂度主要取决于建立索引的文档数量  $n$ ，以及每篇文档的长度（即单词数量，最差情况为  $l$ ）。如果两者都快速增长，那么建立索引的时间会呈现指数级增加。

① 这里假设散列设计得足够好，没有产生冲突。



□ 离线索引的时间和空间复杂度都要远远高于在线查询时的复杂度。

□ 在线查询的时间复杂度主要取决于查询关键词的数量  $k$ ，以及每个关键词命中的文档数（最差情况是  $n$ ）。所以短查询相对于长查询会更快，而包含生僻词的查询也会更快。

随着分析经验的累积，我们会发现时间和空间复杂度往往是可以相互转换的，然后根据实践应用的需求来做平衡。例如，用信息检索建立倒排索引，用分类模型进行建模等，就是典型的通过空间换取时间的案例。设想一下，如果不建立倒排索引，那么可以节省很多空间，但是实时查询的时候，就需要做一次时间复杂度为  $O(n \times l)$  的扫描，这是  $O(k + n)$  的成千上万倍，用户根本无法接受。所以，倒排索引的额外存储开销，会换来查询时间的极大缩短。

最后需要注意的是，这里的时间和空间复杂度只是一种理论上的预估，是根据计算的操作次数，或者是存储的单元个数来统计的。它们假设不同的算法在被衡量的时候都处于同等的环境下，因而不会涉及具体的技术实现和软硬件环境，这和实际的性能指标是有区别的。举个例子，实际应用中，同样是读取 1 000 000 个字符，从内存读取和从硬盘读取的时间完全不一样，但是时间复杂度并不考虑这些差别。类似的，存储 1 000 000 个字符，分别放在簇（Cluster）大小为 64KB 和 4KB 的两种硬盘上，消耗的磁盘空间肯定也不同，空间复杂度同样会忽略这些差别。

无论如何，这种理论上的评估让我们对系统的表现在设计初期就有了大致的了解，提供了方向性的指导。如果在项目一开始没有重视，没有设计出较低的时间和空间复杂度，那么再厉害的编码优化、再强劲的硬件设备也未必能换来良好的性能表现。更糟糕的情况是，若是到了系统即将上线的时候才发现这样的问题，那真是回天乏力了。反之，如果一开始就能选择拥有很低复杂度的方案，那么后面就十拿九稳，颇有运筹帷幄之中，决胜千里之外的成就感。

## 7.2.2 应用系统性能

计算复杂度为性能的预估提供了理论基础，确定了大的方向。但是，系统的性能还会受到实际开发、部署和硬件等因素的影响。所以，我们还需要关注应用系统的实际性能表现，其评估发展至今已经涉及多个方面，包括：

□ 处理能力：计算机处理能力的主要指标有系统基础指标和面向应用的指标。基础指标描述了计算机的各个硬件模块，包括 CPU 使用率、系统负载、内存使用率、磁盘和网络 I/O 等。而面向应用的指标描述了系统处理应用请求的能力，包括响应时间、吞吐量和并发数。通常可以先查看面向应用的指标表现如何，然后再初步判断系统性能是否有明显的问题。如果有问题，再根据基础指标来进一步定位问题。

□ 可靠性：计算机系统正常工作的能力。它要求计算机系统首先是可靠的，或者一旦计算机系统发生了故障，它应该具有容错的能力，再或者系统出错后能迅速恢复。

□ 利用率：即在一段时间内被使用的时间占总时间的百分比，有硬件利用率、软件利

用率、指令利用率等。

□ 功耗及对环境的要求：对于特殊环境下使用的计算机系统尤其重要，如军用、航天计算机、水下计算机等。计算机系统设计人员也需要考虑环境的因素，如电压是否稳定等。

“这么多项目，从哪里入手呢？”

“大宝，别急。考虑到终端用户最为关心的应用层面，我们这里的评估一般都集中在响应时间、吞吐量和并发数上。为了便于你的理解，让我们用生活中的实际案例来解释。对了，你经常去银行办理业务吗？”

“我都是用网上银行。可我爸妈还是需要去的。他们老是抱怨银行里人太多，排队非常浪费时间。”

“好，我们就来看这个例子。”

场景一：下午2点整，大宝的爸爸进入银行，2点30分办理业务结束，离开了银行。如果我们认为进入银行就是银行提供服务的开始，而离开银行是这项服务的结束，那么这个30分钟的时长就被称为服务的“响应时间”。对于计算机系统而言，应用执行一个请求的时间，包括从发出请求开始到最后返回响应数据所需要的时间，就是其响应时间。响应时间是系统最重要的性能指标，直观地反映了系统的“快慢”，关乎用户使用中最直接的感触，对用户体验的影响非常明显，当银行排队人太多时，自然也会引起大宝爸妈的抱怨了。测试的程序通过模拟使用的记录，根据响应和发出请求之间的时间差来计算系统响应时间。可是，记录及获取系统时间这个操作也需要一定的消耗，如果被测试的请求本身需要花费的时间极短，比如几微秒，那么测试程序就无法准确地测试到真实的系统响应时间。在实际运用中，常常采用的办法就是重复请求，记录重复执行上千甚至上万次的时间总和，最后得出每次的平均时间。

场景二：在柜台接待大宝爸爸的是业务员小李，她是银行的老员工，工作经验非常丰富，业务流程也很熟悉。平均每小时能帮助8位客户完成业务。这里平均每小时完成的业务量（8笔）称为服务的“吞吐量”。小张是新来的实习生，对于业务和 workflows 都处于摸索阶段，因此平均每小时只能完成5笔业务。我们就认为小李处理的吞吐量要大于小张。回到计算机系统，吞吐量就是指单位时间内系统处理的请求数量，体现系统的整体处理能力。例如，可以用“请求数/秒”、“处理业务数/小时”或是“访问人数/天”来衡量。对于面向终端顾客的互联网应用来说，因为前端都需要实时响应，所以大多采用“事务数/秒”TPS（Transaction Per Second）来衡量。虽然终端的客户无法看到系统的该项指标，但是这是系统整体处理能力的体现，是性能好坏的关键风向标。

场景三：随着进入银行办理业务的用户越来越多，小李和小张来不及处理，两个服务窗口都排起了长长的队伍。我们将正在办理业务的用户称之为“并发”，而这个人数则称为“并发数”。对于计算机系统来说，并发数就是指系统能够同时处理请求的数目，这个数字也反映了系统的负载特性。虽然高并发可能意味着业务的蓬勃发展，但是理想的情况下，我们是不希望并发数太高的，因为很可能这是系统处理能力达到了瓶颈，成为拖累业务的后腿。

“哦，这么解释我就明白了，非常形象。不过我仍然不太懂为什么不希望高并发的存在？”

“嗯，等我解释完这三者之间的紧密关系，你就能懂了。”

要理解响应时间、吞吐量和并发的关系，同样可以使用银行的案例。当没有人排队，业务窗口没有全部打开的时候（低并发），每位用户都会第一时间得到银行业务员的帮助，业务办理速度非常之快（响应时间短），业务员每小时处理的业务也不多，甚至还可能会出现没有用户接待的情况（吞吐量低）。随着进入银行的人数渐渐增加，每个窗口前都有人在办理（并发适中），用户开始排队等待，业务办理速度放缓（响应时间增长），业务员加快了处理的速度，不停地忙碌，然后增开了服务窗口（吞吐量变高）。如果银行持续涌入大量客户，最终的结局是银行大厅人满为患，每个服务窗口前都有用户（并发很高），人们一边排队一边抱怨（响应时间超长），而银行的服务窗口的处理能力已达到极限（吞吐量无法上升）。从这些逻辑关系，我们也可以推算出三者之间的关系，在并发数一定的情况下，响应时间和吞吐量成反比关系。在并发数持续增加的时候，响应时间和吞吐量二者都开始增加，直至某项先达到瓶颈，那么另外一项就会无限制地增大下去。图 7-8 图示化了三者之间的关系，其中横轴为并发数，纵轴的两条曲线分别为响应时间和吞吐量。

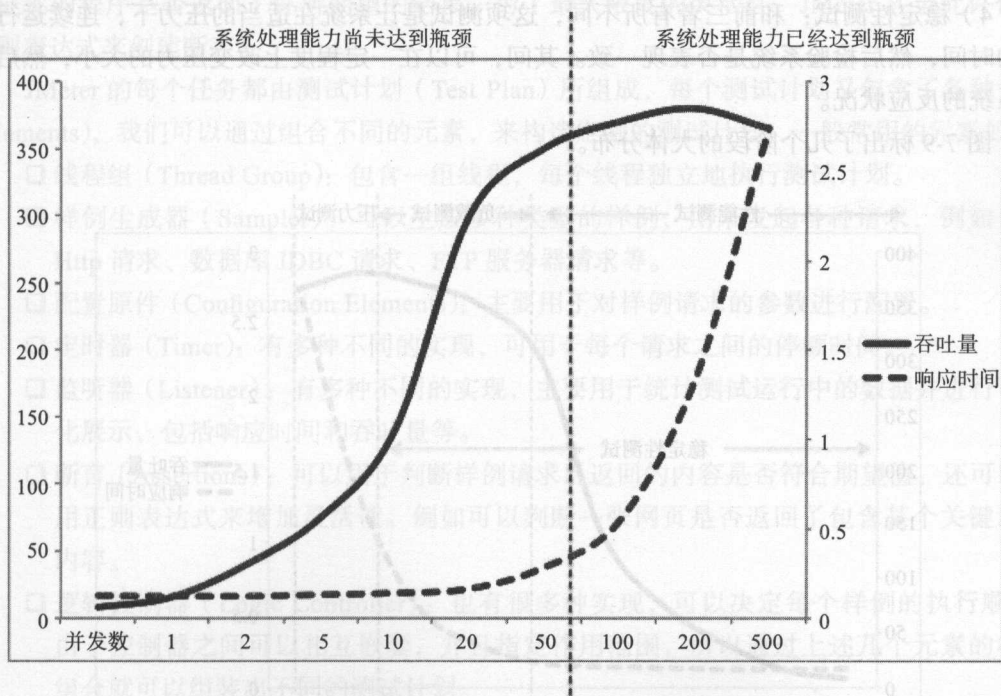


图 7-8 响应时间、吞吐量和并发数三者的关系

最后，三者的关系大致可以近似地用如下公式来表示：



$$CR \propto TPS \times RT$$

其中  $CR$  (Concurrent Request) 表示并发,  $TPS$  (Transaction Per Second) 表示吞吐量,  $RT$  (Response Time) 表示响应时间,  $\propto$  是正比符号。

“哈哈, 是这样的啊。那么对于计算机系统也是同理了, 高并发就要求系统有很强的处理能力, 否则吞吐量上不去, 那么只能无限地放大等待的时间了。对于处理能力弱的系统而言, 高并发确实是个噩梦啊。”

“没错, 你的理解完全正确。”

“那如何发现处理能力的瓶颈呢?”

“可以分为如下几个阶段逐步进行。”

- 1) 性能测试: 以系统设计初期规划的性能指标为预期, 对系统不断施加压力。验证系统在资源可接受的范围内, 是否能达到性能的预期目标。
- 2) 负载测试: 持续增加系统的并发数, 直到系统的某项或多项性能指标达到安全临界, 如果某些资源已经接近或达到饱和状态, 系统的处理能力不但没有提升, 可能还会下降。
- 3) 压力测试: 超过安全临界后, 对系统继续施加压力, 直到系统崩溃、不能再处理任何请求为止, 这样可以获得系统承受压力的极限。
- 4) 稳定性测试: 和前三者有所不同, 这项测试是让系统在适当的压力下, 连续运行较长的时间, 然后检验系统是否表现一致。其间, 可以在一定程度上改变压力的大小, 然后观察系统的反应状况。

图 7-9 标出了几个阶段的大体分布。

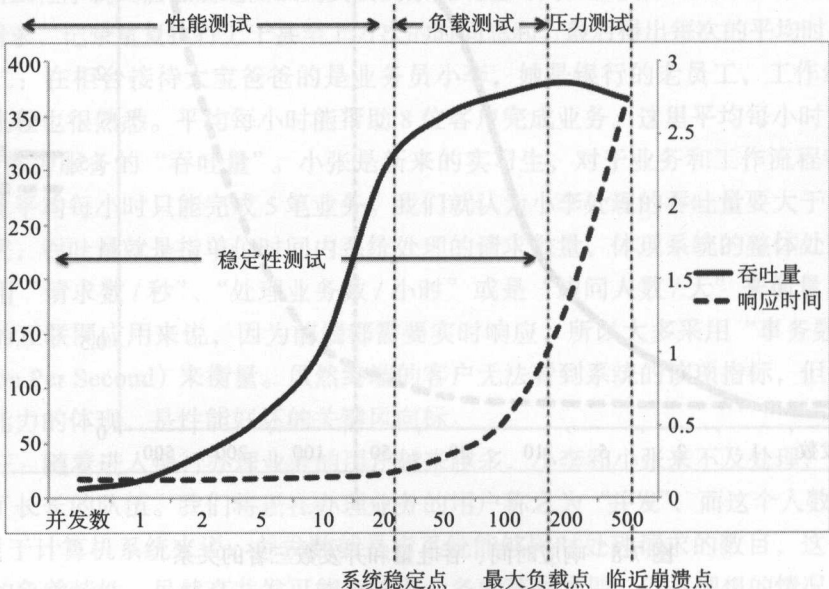


图 7-9 不同的测试阶段

至此，我们理解了应用系统最为重要和常见的性能指标：响应时间、吞吐量和并发数。在了解了这些概念后，也可以实际上手操练一下。下面来看一款开源的性能测试工具：JMeter。

### 7.2.3 JMeter 工具

既然说到了性能的评估，就需要探讨一下相应的测试工具，这里以一款常用的 Apache 开源自动化测试工具——JMeter (<http://jmeter.apache.org>) 为例。先简介下自动化测试，它是把以人为驱动的测试行为转化为机器执行的一种过程，这样既可以节省人力、时间和硬件资源，又可以提高测试效率。由于性能相关的测试需要发送大量的请求，人工完成是不现实的，因此经常会采用自动化的方式。JMeter 目前其最新的可下载版本为 2.13，其最初被设计用于 Web 应用测试，但随着其功能的完善和用户群体的扩大，JMeter 逐步扩展到其他测试领域。它还可以用于测试静态和动态资源，例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库、FTP 服务器等。JMeter 可以用于对服务器、网络或对象模拟出巨大的流量和负载，测试系统在不同压力情况下的稳定性，并产生报表和图形来协助你分析整体的性能。另外，JMeter 还能够对应用程序做功能上的回归测试，通过创建带有断言的脚本来验证你的程序是否返回了你所期望的结果。为了最大限度的灵活性，JMeter 甚至允许使用正则表达式来创建断言。

JMeter 的每个任务都由测试计划 (Test Plan) 所组成，每个测试计划又包含了各种元素 (Elements)，我们可以通过组合不同的元素，来构造定制的测试计划。一般常用的元素如下。

- ❑ 线程组 (Thread Group)：包含一组线程，每个线程独立地执行测试计划。
- ❑ 样例生成器 (Sampler)：可以生成多种类型的样例，用来发起各种请求，例如 Web Http 请求、数据库 JDBC 请求、FTP 服务器请求等。
- ❑ 配置原件 (Configuration Elements)：主要用于对样例请求的参数进行配置。
- ❑ 定时器 (Timer)：有多种不同的实现，可用于每个请求之间的停顿时间。
- ❑ 监听器 (Listener)：有多种不同的实现，主要用于统计测试运行中的数据并进行可视化展示，包括响应时间和吞吐量等。
- ❑ 断言 (Assertions)：可以用于判断样例请求后返回的内容是否符合期望值，还可以使用正则表达式来增加灵活性。例如可以判断一张网页是否返回了包含某个关键词的内容。
- ❑ 逻辑控制器 (Logic Controller)：也有很多种实现，可以决定每个样例的执行顺序。由于控制器之间可以相互嵌套，并且指定作用范围，所以通过上述几个元素的相互组合就可以组装出不同的测试计划。

接下来看一个实践的例子，运行 JMeter，在主界面首先建立测试计划 Test，如图 7-10 所示。

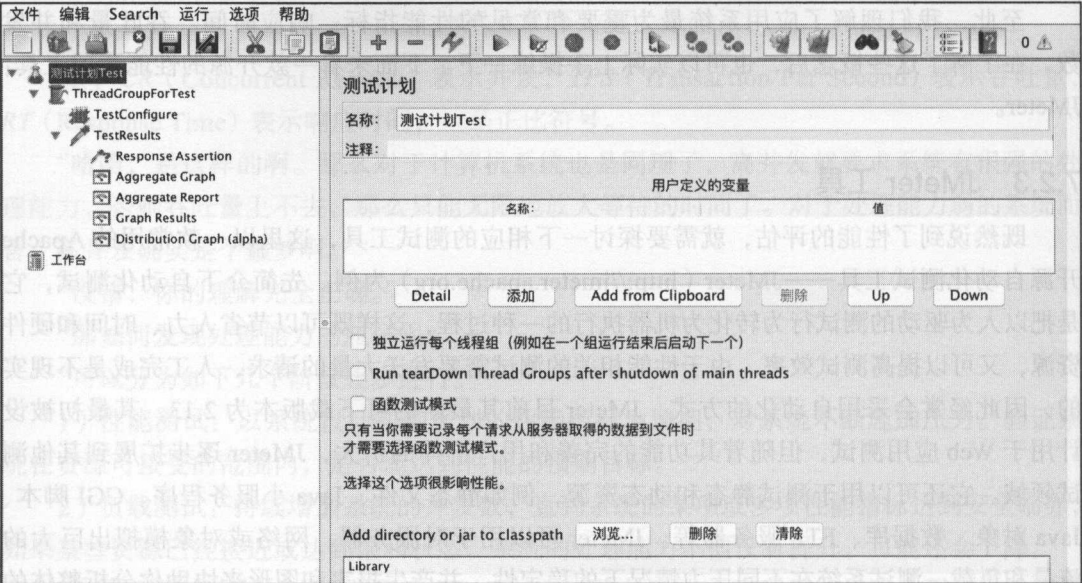


图 7-10 建立测试计划

然后在测试计划 Test 中增加线程组，如图 7-11 所示。常用的配置是线程数，也可以认为是并发数，这里设置为 5。还有循环次数设置为 300 次。

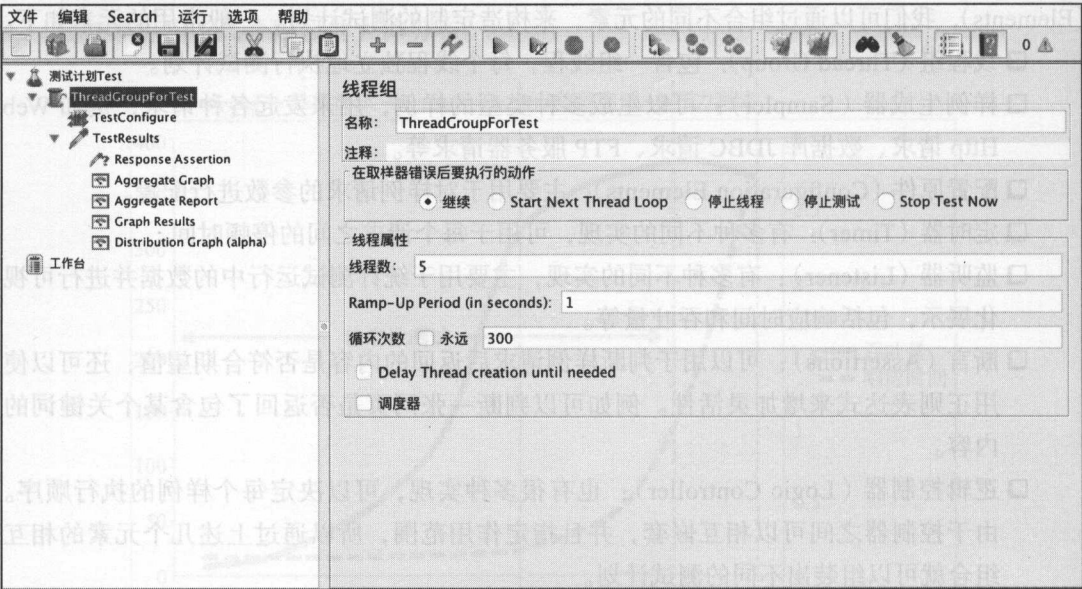


图 7-11 添加线程组

接着，在线程组 ThreadGroupForTest 中增加配置元素，如图 7-12 所示。常用的配置是



待测服务器的 IP 地址和端口号。这里 HTTP 请求的路径将放在样例生成器中进行配置。

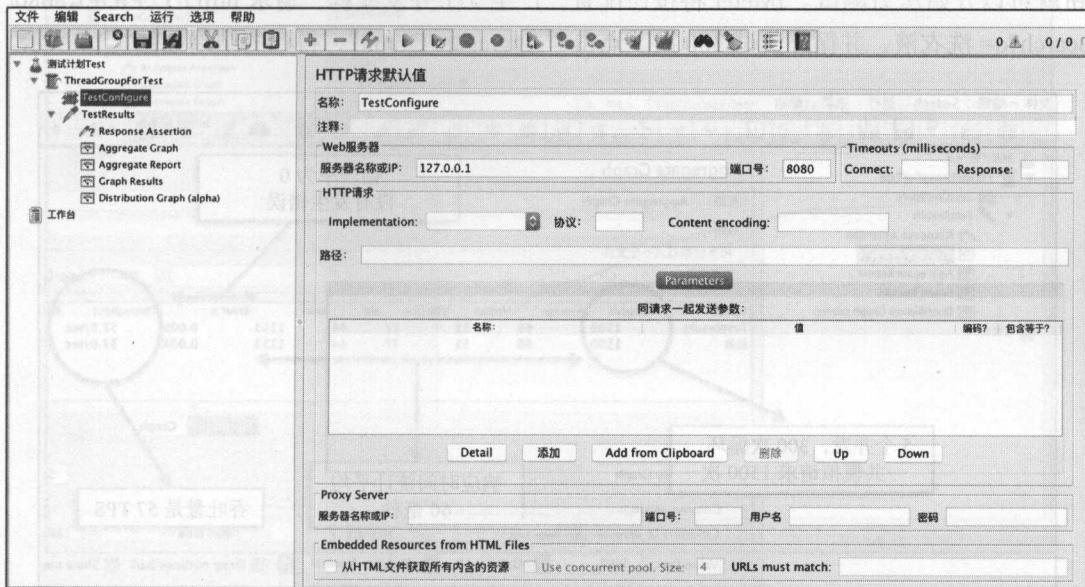


图 7-12 添加配置元素

在线程组 ThreadGroupForTest 中增加样例生成器，如图 7-13 所示。常用的配置是 HTTP 请求路径，这里是 /search?q=洗衣液，以及请求方式，这里是 POST。

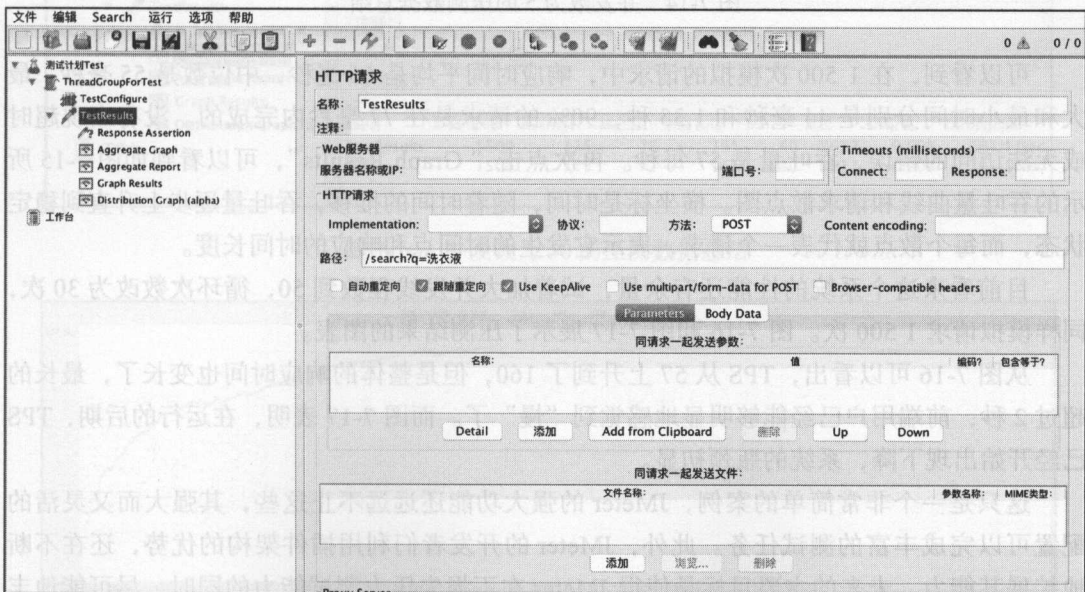


图 7-13 添加样例生成器

这样，我们就完成了一个最基本的测试计划，点击工具栏上绿色三角形的“启动”按钮就可以开始压力测试。JMeter 将按照配置，产生 5 个并发线程，请求 `http://127.0.0.1:8080/search?q=洗衣液`，并循环测试 300 次。点击“Aggregate Graph”，得到的结果如图 7-14 所示。

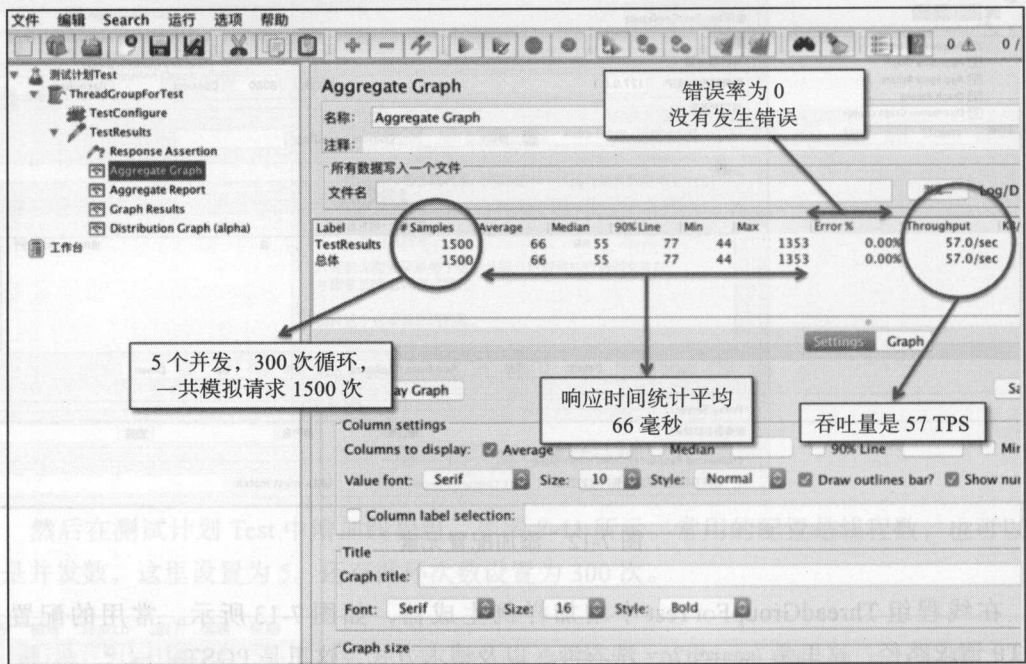


图 7-14 并发数为 5 的压测数据总结

可以看到，在 1 500 次模拟的请求中，响应时间平均是 66 毫秒，中位数是 55 毫秒，最大和最小时间分别是 44 毫秒和 1.35 秒，90% 的请求是在 77 毫秒内完成的。没有出现超时或无法访问的错误，吞吐量是 57 每秒。再次点击“Graph Results”，可以看到如图 7-15 所示的吞吐量曲线和请求散点图。横坐标是时间，随着时间的推移，吞吐量逐步上升直到稳定状态，而每个散点就代表一个请求，表示它发生的时间点和响应的长度。

目前看来这个系统的性能还有余量，试着加大并发线程数到 50，循环次数改为 30 次，同样模拟请求 1 500 次。图 7-16 和图 7-17 展示了压测结果的图表。

从图 7-16 可以看出，TPS 从 57 上升到了 160，但是整体的响应时间也变长了，最长的超过 2 秒，前端用户已经能够明显地感觉到“慢”了。而图 7-17 表明，在运行的后期，TPS 已经开始出现下降，系统的瓶颈初显。

这只是一个非常简单的案例，JMeter 的强大功能还远远不止这些，其强大而又灵活的配置可以完成丰富的测试任务。此外，JMeter 的开发者们利用插件架构的优势，还在不断地扩展其能力。未来的主要目标是使得 JMeter 在不损失压力测试能力的同时，尽可能地丰富功能和回归测试的特性。

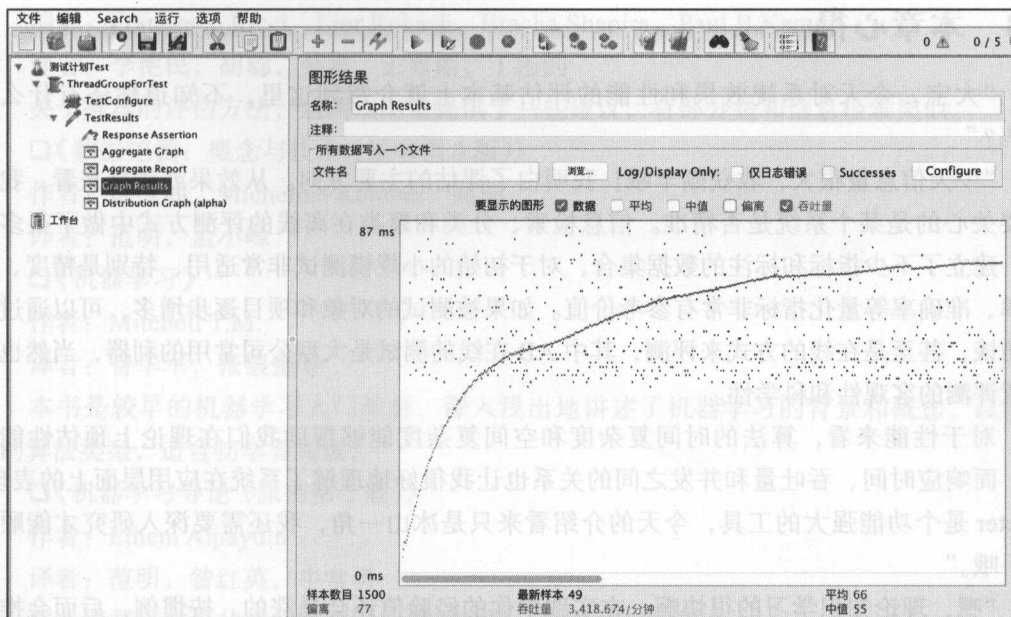


图 7-15 并发数为 5 的压测结果可视化

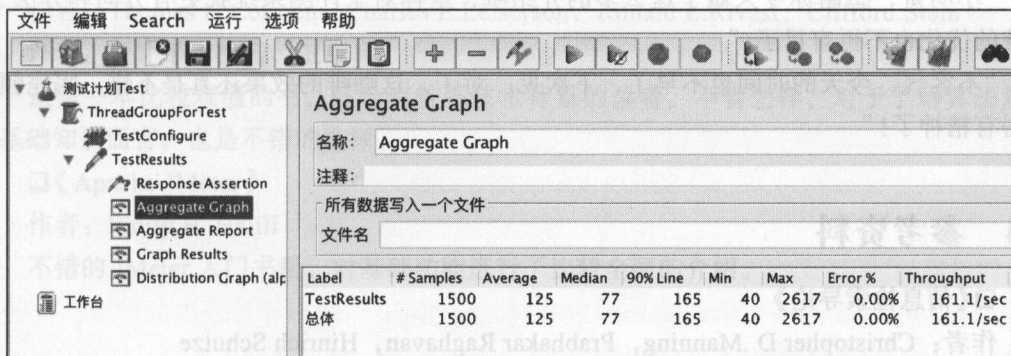


图 7-16 并发数为 50 的压测数据总结

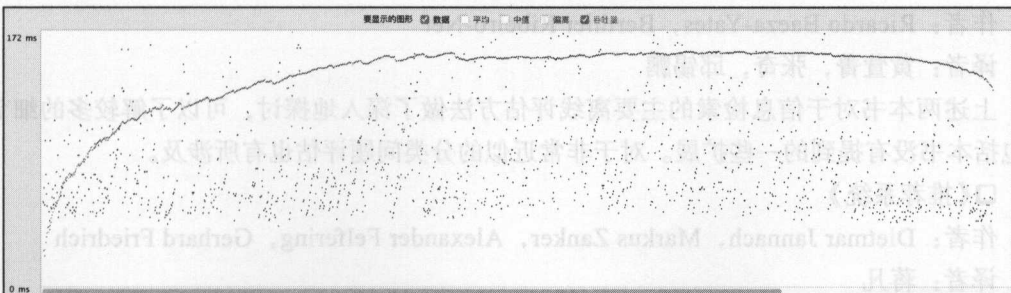


图 7-17 并发数为 50 的压测结果可视化



## 7.3 本章心得

“大宝，今天对系统效果和性能的评估基本上就介绍到这里，不知道你还有什么疑问吗？”

“今天信息量很大，收获颇丰哦！我明白了评估的主要方向。从效果或质量来看，我们需要关心的是某个系统是否精准。信息检索、分类和聚类在离线的评测方式中做了很多尝试，建立了不少指标和标注的数据集合，对于初始的小规模测试非常适用，特别是精度、召回率、准确率等量化指标非常有参考价值。如果被测试的对象和项目逐步增多，可以通过用户访谈，甚至是在线的方式来评测，其中 AB 在线的测试是大型公司常用的利器，当然也要注意评测的客观性和科学性。

对于性能来看，算法的时间复杂度和空间复杂度能够帮助我们在理论上预估性能表现，而响应时间、吞吐量和并发之间的关系也让我很好地理解了系统在应用层面上的表现。JMeter 是个功能强大的工具，今天的介绍看来只是冰山一角，我还需要深入研究才能顺利上手哦。”

“嗯，理论知识学习的很快啊。在实践中你的经验值还会暴涨的。按惯例，后面会推荐一些参考资料。”

“小明哥，感谢你今天做了这么多的介绍哦，这样对于评测系统我更有方向和方法了，系统的优化也不再盲目了。”

“不客气，今天的时间也不早了，下次见。对了，这咖啡的效果还真是不错，喝完顿时觉得有精神了！”

## 7.4 参考资料

### □《信息检索导论》

作者：Christopher D .Manning, Prabhakar Raghavan, Hinrich Schutze

译者：王斌

### □《现代信息检索（原书第2版）》

作者：Ricardo Baeza-Yates, Berthier Ribeiro-Net

译者：黄萱菁，张奇，邱锡鹏

上述两本书对于信息检索的主要离线评估方法做了深入地探讨，可以了解较多的细节。还包括本书没有提到的一些扩展。对于非常近似的分类问题评估也有所涉及。

### □《推荐系统》

作者：Dietmar Jannach, Markus Zanker, Alexander Felfering, Gerhard Friedrich

译者：蒋凡

### □《推荐系统：技术、评估及高效算法》

编者: Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B.Kantor

译者: 李艳民, 胡聪, 吴宾, 王雪丽, 丁彬钊

关于推荐的评估方法, 这两本书也提供了一些建议, 有部分是和信息检索类似。

#### □《数据挖掘: 概念与技术 (原书第3版)》

作者: 韩家炜, Micheline Kamber, 裴健

译者: 范明, 孟小峰

#### □《机器学习》

作者: Mitchell T.M.

译者: 曾华军, 张银奎等

本书是较早的机器学习入门教材, 深入浅出地讲述了机器学习的背景和概念, 以及常用的算法类型, 适合初学者阅读。

#### □《机器学习导论 (原书第2版)》

作者: Ethem Alpaydin

译者: 范明, 咎红英, 牛常勇

这些书都探讨了分类、聚类等问题的评测, 尤其是离线部分。

#### □《算法导论 (原书第3版)》

作者: Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein

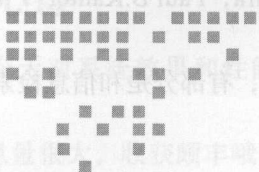
译者: 殷建平, 徐云, 王刚等

这是一本比较难懂的书, 适合有较强技术背景的读者。不管怎样, 对于了解算法复杂度基础知识而言, 也是不错的选择。

#### □《Apache JMeter》

作者: Emily H. Halili

不错的 JMeter 入门书籍, 对基础功能进行了比较全面的介绍。



转眼之间，大宝跟随小明学习大数据已经有半年了。这天，当他们再次相遇时，大宝提出了这样的一个问题。

“小明哥，前面你已经先后介绍了数据的获取、存储和处理环节，也阐述了信息检索和数据挖掘这样的高级技术。有了你这许多的教诲和指导，最近我大数据方面的知识水平是突飞猛进啊。不过，如果说利用这些知识去解决企业的实际业务需求，我感觉还欠缺点整体感，不是很系统。”

“嗯，你的直觉是正确的。在解决实际的业务需求时，通常只采用一两项技术是远远不够的。我们需要将它们有机地结合起来。这里先画个草图，解释一下通常情况下是如何组织这些技术点的。”

说着，小明拿起纸笔唰唰唰地在纸上画出了图 8-1。最底层是数据的收集，可通过 Nutch 这样的爬虫系统，从互联网上采集我们所需的数据。例如，时事新闻、行业动态、竞争对手信息等。而公司内部产生的数据则可以通过 Flume 等系统进行采集。如果我们建立了一个互联网站点，那么用户的访问行为是需要跟踪和分析的，Flume 就可以将访问的日志定期地传送并保存到分布式存储中。

数据到了存储层后，Hadoop 的 HDFS 提供了最基本的持久化分布式文件系统，对于数据查询和处理要求不高的可以直接使用它来存放，例如行业最近一年的重大新闻集合。对于高级应用的开发者而言，HBase 和 MongoDB 则提供了类似关系型数据库的功能。同时，HBase 的列式存储更便于数据定义的随时更改，而 MongoDB 的嵌入式文档则支持复杂的层级结构，这些都为欠缺规范的大规模数据提供了更高的灵活性。这也意味着，刚开始我们并不一定要严格地定义用户访问日志的格式，而是随着应用需求的不断更新而变化。而支持非持久化的系统，包括 Redis、Berkeley DB 和 Memcached，则为前述的存储数据库提供了缓



存机制，可以大幅地提升系统的响应速度，降低持久化存储的压力。

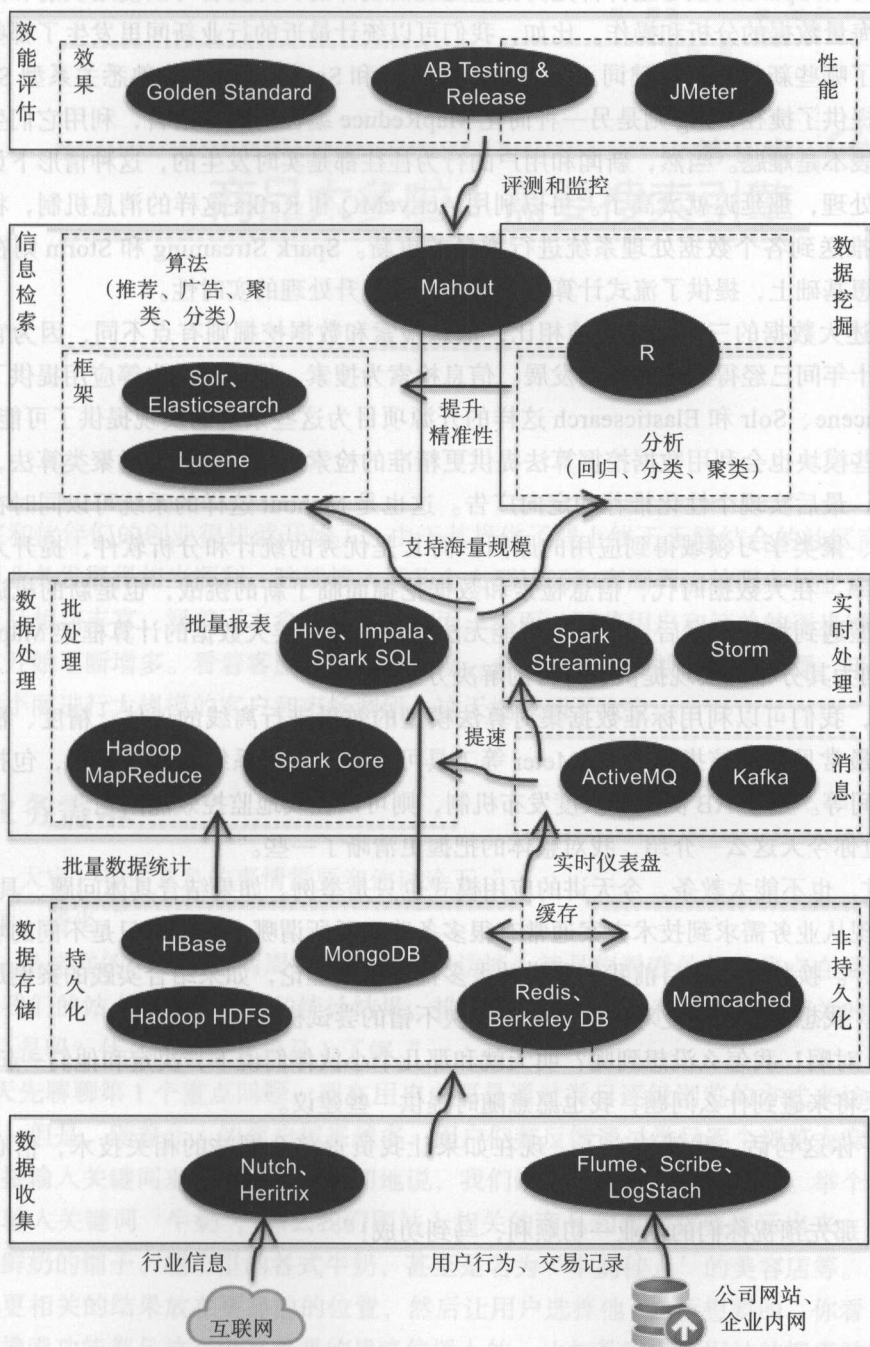


图 8-1 大数据的主要技术点及其相互关系

当读写数据不再是问题时，就要考虑如何利用它们产生更大的价值了。Hadoop 的 MapReduce 和 Spark 的核心组件都是为批量处理而设计的，开发者可以使用映射和归约的思想，进行海量数据的分析和操作。比如，我们可以统计最近的行业新闻里发生了哪些重大事件，产生了哪些新的热门关键词。而 Hive、Impala 和 Spark SQL 则为熟悉关系型 SQL 语言的使用者提供了捷径，Pig 则是另一种简化 MapReduce 编程的脚本语言，利用它们生成基本的统计报表不是难题。当然，新闻和用户的行为往往都是实时发生的，这种情形下如果总是等着批量处理，那延迟就太高了。可以利用 ActiveMQ 和 Kafka 这样的消息机制，将数据的变化及时推送到各个数据处理系统进行增量的更新。Spark Streaming 和 Storm 则在映射和归约的思想基础上，提供了流式计算框架，进一步提升处理的实时性。

与上述大数据的三个基础设施相比，信息检索和数据挖掘则有点不同，因为它们在过去的二三十年间已经得到了充分的发展。信息检索为搜索、推荐和广告等应用提供了理论基础，而 Lucene、Solr 和 Elasticsearch 这样的开源项目为这些系统的实现提供了可能。此外，人们在某些模块也会利用数据挖掘算法提供更精准的检索效果，例如通过聚类算法，对用户进行分群，最后实现个性化推荐和定向广告。这也是 Mahout 这样的系统可以同时推荐领域和分类、聚类学习领域得到应用的原因。而 R 是优秀的统计和分析软件，提升人们运用模型的效率。在大数据时代，信息检索和数据挖掘面临了新的挑战，也是新的机遇。传统的理论模型遇到海量数据后，单机很可能无法应付。这时候大数据的计算框架 MapReduce、Spark 等则为其分布式实现提供了可行的解决方案。

最后，我们可以利用标准数据集对算法模型的效果进行离线的评估，精度、覆盖率和准确率是最常见的考核指标。而 JMeter 等工具可以测试应用系统的性能表现，包括吞吐量和响应时间等。至于 AB 测试和灰度发布机制，则可以在线地监控效能问题。

“经过你今天这么一介绍，我对整体的把握更清晰了一些。”

“不过，也不能太教条，今天讲的应用模式也只是举例。如果结合具体问题、具体分析，我们会发现从业务需求到技术方案通常有很多条路，无所谓哪个最优，只是不同的场景下有不同的设计。换句话说，目前我们所讲的大多都是限于理论，如果结合实践的案例那自然效果倍增啊。我想，这次创业对你而言也是一次不错的尝试机会哦！”

“哦，对啊！我怎么没想到呢？明天就和那几个小伙伴们说下，决定和他们一起干了！”

“如果将来碰到什么问题，我也愿意随时提供一些建议。”

“有了你这句话，我就放心了。现在如果让我负责整个网站的相关技术，信心那是满满的。”

“好，那先预祝你们的创业一切顺利，马到成功！”

## 商品太多啦！需要搜索引擎

### 9.1 业务需求

“嗨，大宝，有件紧急的事情需要和你讨论下。”

“小丽，请坐。”

“嗯，是这样的。我们这两周一直在做用户访谈，就是想看看他们的痛点在哪里，为什么要抱怨我们的站点。根据最新的统计结果，排名前两位的都是和技术系统相关的问题。”

“哦，是吗？什么问题？我想马上了解。”

“今天先聊聊第1个重点问题。现在用户主要是通过类目逐级浏览的方式来访问周边社区商品的。但是，随着介入的商户越来越多，用户们都反馈通过类目逐个浏览太麻烦了，没有办法直接输入关键词来查找商品。确切地说，我们缺一个站内的搜索功能。举个例子，当一个顾客输入关键词‘牛奶’，那么我们网站上相关的商品和商铺都要展示出来，包括社区门口售卖鲜奶的铺子、超市里的各式牛奶，甚至是名为‘牛奶佳人’的美容店等。当然，我们需要将更相关的结果放在更靠前的位置，然后让用户选择他/她所想要的。你看，各大电商网站的搜索功能都是放在极其重要的战略位置上的。比如某家电商网站的搜索结果页就是如此。”说着，小丽打开电脑相关网页指给我看（见图9-1）。“你看，左侧有分类选择，上



面还有用于导购的筛选项，包括品牌、功效、是否进口等。还有好多排序选型，包括综合、销量、价格等。”

“好的，这确实是个明显的问题，我也完全明白搜索系统对于公司的价值。这样，我们团队立即安排搜索功能的设计和开发。小丽，谢谢你的及时反馈！”



图 9-1 搜索结果示意图

## 9.2 产品设计和选型

送走了小丽，大宝立即找来几个开发骨干开始谋划搭建公司的第一个搜索引擎。有人提议用数据库的 SQL 语言来实现。大宝摇了摇头，因为现在他的心里很清楚，这种场景下数据库是很难满足用户多方位的需求的，这主要是因为数据库有如下几个特性。

- 精确的匹配方式：数据库擅长的是精准匹配，例如，根据商品的 ID 或完整的标题来查找。而网站的顾客是无法输入商品的条形码或唯一标识 ID 查找的，他们一定是输

入某些简短的关键词，例如“牛奶”、“川菜”、“美甲”等。即使某些数据支持模糊匹配，例如 SQL 语言中的 Like 语法，其处理和匹配的效率也相对较低下，很难满足高并发的网站流量。

❑ 相关性考虑不足：搜索引擎返回的结果和用户的输入是否相关，是决定搜索功能成功与否的关键。试想一下，在客户输入“牛奶”后，返回的都是牛奶味饼干，甚至是牛奶色的衣服，那么用户的体验该是多么的糟糕。普通的数据库是完全不会考虑这些层面的需求的。

❑ 较高的系统耦合度：如果使用数据库做搜索，那么对于数据库的使用一般有两种选择。第一，直接读取主库<sup>①</sup>，第二，读取备库<sup>②</sup>。选择主库的好处是节省存储空间，而且可以获得最新的商品信息。但是风险在于搜索请求的流量很大，可能会对主库造成过大的压力，甚至导致宕机，那么整个核心系统就无法注册、登录和交易了，这对公司的业务是致命性的打击，因此并不推荐。这种情况我们称之为系统耦合度太高，不利于开发、维护和事故处理。

❑ 冗余的系统：当然，如果不使用主库，还可以使用备库来做搜索。这样在很大程度上可以缓解主库的压力。但是，备库通常是直接将整个主库全盘复制，这会导致更多的数据存储消耗。但搜索的时候并非所有的数据表和字段都要被使用到，因此造成了浪费。如果要更高效地利用存储资源，还需要更精细化的主从同步配置，而且也未必能细分到字段级别。

❑ 缺乏高级功能：目前，用户已经习惯在浏览搜索结果的同时，看到各种导购属性和细分品类，以便进行进一步的筛选，直至选中满意的商品为止。而数据库缺少对这种应用需求的足够支持，要是完全靠自己重新编写不仅耗费时间，而且效果和性能也都无法得到保证。

❑ 没有很好的辅助模块：对于一个成熟的搜索引擎而言，通常需要多个模块协同工作。例如，中文网站需要一个精准的中文分词处理模块，将“精致的美甲套餐”整个字符串切分为“精致”、“的”、“美甲”和“套餐”等若干中文关键词。大宝的公司也有同样的需求，但是数据库一般都没有集成这类模块。

考虑到这些情况，再结合第5章的指导，大宝凭直觉意识到应用 Lucene 和 Solr、Elasticsearch 系统的时候到了。首先，从理论上说，这些开源系统都秉承了信息检索理论的搜索基础，提供了倒排索引的机制，不仅支持模糊匹配，还能保证查询的效率。各种相关性的模型也能让开发者更有目的地调整实现的方向和细节。其次，从开发实现的角度来看，独立的搜索系统在充分减轻数据库压力的同时，也可以避免不必要的数据冗余。Lucene 的高版本、Solr 和 Elasticsearch 都已经支持切面 (Facet) 和聚合 (Grouping)，对于实现导购筛选

① 最主要的数据库，新的数据都在这里写入和更新。对于电商等互联网系统，这里存储的是最为核心的交易和用户数据。

② 备库会定期从主库同步数据，因此更新有一定的延迟，但是可以为主库分担压力。

这样的需求非常合适，集成各种开源的分词软件也很方便。

确定基本思路后，大家又开始讨论应该使用 Lucene 还是 Solr，或是 Elasticsearch。由于 Lucene 相对底层，如果采用它来搭建，灵活性肯定是最高的。不过，其工作量也会是比较大的，很多逻辑需要通过自己研发来实现封装。而 Solr 和 Elasticsearch 的友好度更高，通过配置就能实现大部分的基本搜索需求，常见的定制化也可以通过修改源码达到要求，总体上来说是最好的选择。考虑到对于初学者来说，Solr 的入门更快，因此大宝的团队最终决定使用 Solr 来搭建公司的商品和商铺搜索。

大宝还及时地想到了 Solr 的一个重要功能：DIH（Data Import Handler）数据导入模块，它可以将多种外在的数据源直接导入到 Solr，然后直接进行索引。只要是提供了 JDBC（Java DataBase Connectivity）的数据库都可以和 DIH 配合工作，例如 Oracle、MySQL、微软的 SQL Server 等。开发者只需要提供数据库连接的参数，还有 SQL 查询语句，DIH 就能自动查询数据库，然后将结果集合转化为 Solr 中的文档来索引，同时还支持全量和增量的更新。这非常适合公司的现状，兄弟们不用再进行额外的编码来将数据从 MySQL 里导入到 Solr 中。

由于商品的数量及用户的访问流量都呈现快速增长的趋势，因此从系统架构的角度考虑，可以一开始就充分利用 Solr 的分布式特性，为系统的水平性扩展做好准备。Solr 在 4.0 版本之前，其分布式系统是通过 Master-Slave 架构实现的，由专门的主服务器（Master）来转发索引更新的请求，然后由多个从服务器（Slave）来响应搜索查询的请求。这种实现方案虽然简单，但是却存在单点故障的风险，万一主服务器宕机，那么整个集群都无法更新。从 4.0 开始，Solr 提供了一个新的分布式架构——SolrCloud，它可利用 ZooKeeper 来管理机器中的机器节点。ZooKeeper 会替集群选出一位 leader 角色进行更新或查询请求的分发。若 leader 宕机，ZooKeeper 就会从剩余还存活的机器中再次自动选举出新的 leader，继续完成索引的更新和搜索查询，这就消除了单点故障的隐患。综合考虑，这里选择的分布式架构也是 SolrCloud。

因此，其基本架构如图 9-2 所示，Solr DIH 负责解读配置好的 SQL 语言，及时获取数据库中的数据变化，并推送到 SolrCloud 里，然后通过 ZooKeeper 来管理，从 DIH 模块接收的数据变化会自动传播到所有服务器节点，保证索引同步更新。前端应用需要搜索的时候，发送请求到 SolrCloud，集群也会负责挑选服务器节点进行响应，并返回搜索结果。因为需要处理中文，这里引入了开源的分词软件 IKAnalyzer（<http://code.google.com/p/ik-analyzer/>）。IKAnalyzer 是一个开源的、基于 Java 语言开发的轻量级中文分词工具包，结合了词典分词和文法分析算法，可以方便地和 Solr 集成，用于索引字段和查询输入的中文切词。

这里的前端应用和 SolrCloud 之间没有引入接口封装的 API 层，是因为目前的业务需求还比较简单，前端开发者只需要理解最基本的 Solr 查询语法，就能完成任务。如果将来业务场景变得越来越复杂，那么就需要专门的团队来负责封装和维护一套 API，进行面向服务



的应用开发 SOA(Service Oriented Application)，后面的章节也会介绍 SOA 的原理和必要性。

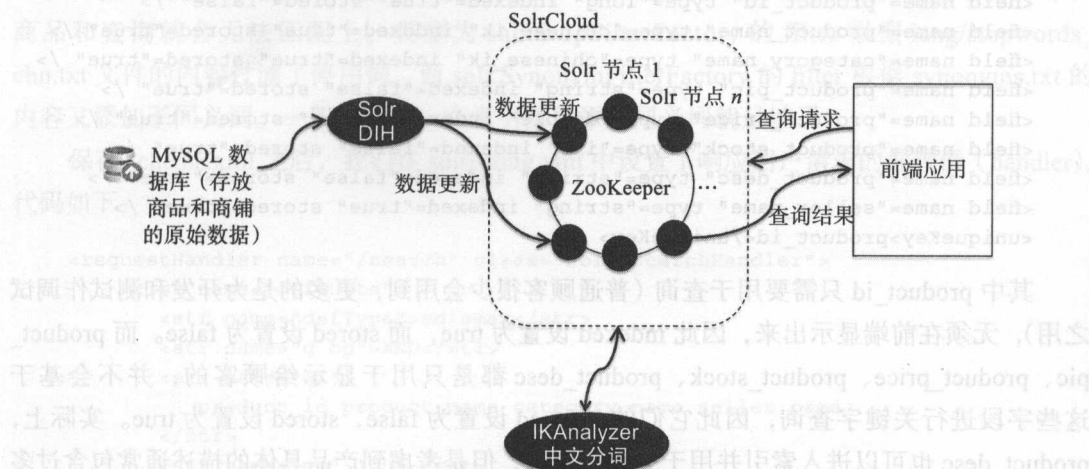


图 9-2 使用 Solr DIH 和 SolrCloud 搭建的基本框架

## 9.3 实现方案

### 9.3.1 数据定义和配置

本次技术方案的最终定稿是让社区周边接入商家的商品全部进入搜索引擎，这里商品的定义是广义的，既包含牛奶、咖啡、手机这样的实物，也包含美甲、按摩、电影这样的服务类和娱乐类项目。每个商品（product）定义的字段如表 9-1 所示。

表 9-1 商品字段的定义

名称	含义	类型
product_id	商品唯一的 ID	long
product_name	商品名称，例如“牛奶”、“美甲套餐”、“寻龙诀”等	string
category_name	商品分类名称，例如“超市”、“餐饮”、“美容”、“电影”。当这些名称进入索引后，就可以保证用户在搜索“电影”这种宽泛词的时候，能够看到这些分类里的产品	string
product_pic	商品图片链接	string
product_price	商品的价格	double
product_stock	商品的库存	int
product_desc	商名的具体描述，例如产品介绍、规格特性和服务条款	string
seller_name	商家的名称，例如超市、美容院、电影院的名字	string

打开并编辑 Solr<sup>①</sup> 中的 schema.xml 文件，加入如下字段配置：

```
<field name="product_id" type="long" indexed="true" stored="false" />
<field name="product_name" type="chinese_ik" indexed="true" stored="true" />
<field name="category_name" type="chinese_ik" indexed="true" stored="true" />
<field name="product_pic" type="string" indexed="false" stored="true" />
<field name="product_price" type="double" indexed="false" stored="true" />
<field name="product_stock" type="int" indexed="false" stored="true" />
<field name="product_desc" type="string" indexed="false" stored="true" />
<field name="seller_name" type="string" indexed="true" stored="true" />
<uniqueKey>product_id</uniqueKey>
```

其中 product\_id 只需要用于查询（普通顾客很少会用到，更多的是为开发和测试作调试之用），无须在前端显示出来，因此 indexed 设置为 true，而 stored 设置为 false。而 product\_pic、product\_price、product\_stock、product\_desc 都是只用于显示给顾客的，并不会基于这些字段进行关键字查询，因此它们的 indexed 设置为 false，stored 设置为 true。实际上，product\_desc 也可以进入索引并用于关键字查询，但是考虑到产品具体的描述通常包含过多的信息，很可能会导致不相关的搜索结果排在前面，影响用户的体验。因此这里做了简化处理，暂不对描述信息进行索引，以确保搜索的精准性。此外，product\_name、category\_name 和 seller\_name 都是既要进行索引，也要进行展示的字段，因此 indexed 和 stored 都设置为 true。略有不同的是，product\_name 和 category\_name 需要进行中文分词，因此需要采用稍后介绍的 IK 分词模块，而 seller\_name 没有进行分词，必须完整命中商家的名称，确保不会产生歧义。最后，用 uniqueKey 来表示值必须是唯一的字段。

同样在 schema.xml 文件中，加入了分词组件、停用词（stopword）和同义词过滤的配置，代码如下：

```
<fieldType name="text_chinese" class="solr.TextField">
  <analyzer type="index" class="org.wltea.analyzer.lucene.IKAnalyzer">
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_chn.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false" />
  </analyzer>
  <analyzer type="query" class="org.wltea.analyzer.lucene.IKAnalyzer">
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_en.txt" />
  </analyzer>
</fieldType>
```

名称为 text\_chinese 的 fieldType 是用于 product\_name 和 category\_name 的分词，type 为 index 的 analyzer，表示用于索引阶段的切分，而 type 为 query 的 analyzer，表示用于查询阶段的切分。一般索引和查询两个阶段的分词配置要相同，否则可能会导致无法匹配的尴

① 后面的 Solr 实践部分都是以 Solr 4.8.1 版本为例的。

尬。例如，建立索引的时候将商品标题里的“手机保护壳”切分为“手机”、“保护”和“壳”，但是查询时又将用户输入的“手机保护壳”切分为了“手机”和“保护壳”，那么商品和查询就会无法匹配上。类型为 solr.StopFilterFactory 的 filter 根据 lang/stopwords\_chn.txt 文件的内容过滤了停用词，而 solr.SynonymFilterFactory 的 filter 根据 synonyms.txt 的内容又添加了同义词。一般情况下，在索引阶段添加同义词就足够了。

保存 schema.xml 之后，我们在 solrconfig.xml 中设置了响应用户请求的处理器（handler），代码如下：

```
<requestHandler name="/search" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="defType">edismax</str>
    <str name="q.op">AND</str>
    <str name="qf">
      product_id product_name category_name seller_name
    </str>
    <str name="mm">100%</str>
  </requestHandler>
```

在这个名为“search”的处理器中，将 defType 设置为 edismax，表示系统会使用 edismax 方式解析用户的输入，将 q.op 设置为 AND 表示关键词之间默认的是“与”的关系，也就是期望返回的商品中输入的每个关键词都要出现，qf 的内容表示需要进行关键词查询的字段，这里依次是 product\_id、product\_name、category\_name 和 seller\_name，用户输入的关键词会在这 4 个字段中进行搜寻，而 mm 则表示某个商品中至少要出现多少比例的关键词才能返回。当然，solrconfig.xml 还有很多其他方面的配置，包括 DIH 相关的，以及性能调优参数等。

### 9.3.2 集群搭建

配置完单个服务节点的 schema.xml 和 solrconfig.xml 之后，就可以启动 Solr Cloud 的集群了。首先在第一个节点的终端输入如下命令：

```
nohup java -Dcollection.configName=product -Dbootstrap_confdir=./solr/
product/conf -DzkRun -DnumShards=3 -DreplicationFactor=2 -server
-XX:+UseConcMarkSweepGC -XX:+UseParNewGC -jar start.jar &
```

该命令除了启动 Solr 的服务以外，还会将本地的 schema.xml 和 solrconfig.xml 等配置都上传到 ZooKeeper，让其完成配置内容的共享。-Dcollection.configName=product 指定了 ZooKeeper 上这些上传配置的名称，-Dbootstrap\_confdir=./solr/product/conf 指定了需要上传的配置文件所在的路径。-DzkRun 表示使用 Solr 自带的 ZooKeeper，更容易上手。-DnumShards=3 表示索引切片数量为 3，-DreplicationFactor=2 表示每个索引切片的副本数量为 2。剩下的基本都是 JVM 内存调优的参数。如果启动无误，打开 Solr 管理界面中左侧的 Cloud 选项，将会看到如图 9-3 所示的界面。



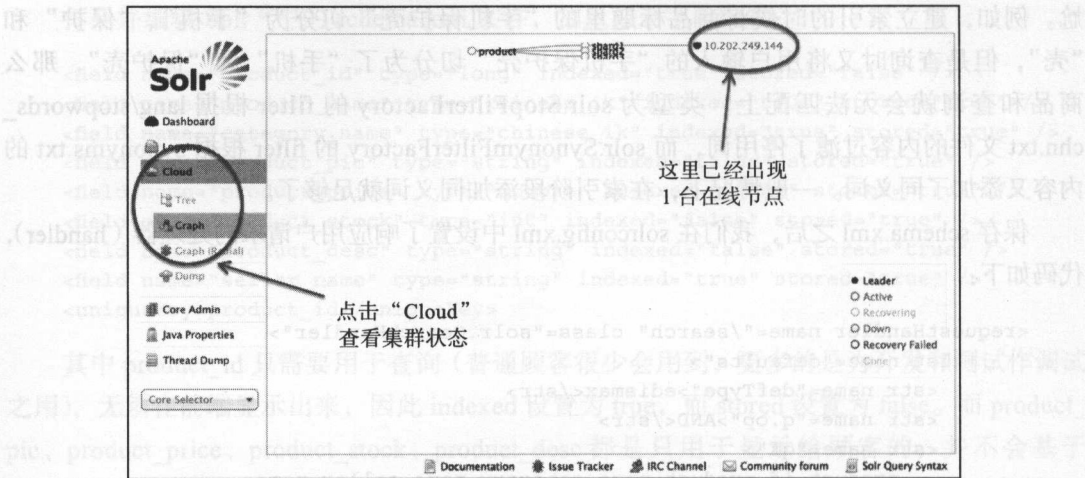


图 9-3 第一台 Solr 服务启动成功，显示 3 个切片，一个节点在线

然后在其他机器节点上输入下列命令以启动剩余的 Solr 服务：

```
nohup java -DzkHost=10.202.249.144:9983,10.202.249.145:9983, 10.202.249.146:9983,
10.202.249.147:9983,10.202.249.148:9983 -DnumShards=3 -DreplicationFactor =2
-server -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -jar start.jar &
```

注意，这里无须再上传配置文件，只需要连接各个节点的 ZooKeeper 客户端端口，其中的 9983 是 Solr 自带的 ZooKeeper 的默认端口。剩余配置类似第一台。再次启动 4 个节点后，图 9-4 显示了集群的现状。由于节点数量不够，因此第三个切片只部署在了 1 个节点上，其他切片则部署在 2 个节点上。另外，值得注意的是，要修改 core.properties 文件中的 coreNodeName，要确保它在各个节点中都不会重复，否则会导致集群节点的冲突，启动失败。

```
name=product
config=solrconfig.xml
schema=schema.xml
dataDir=data
coreNodeName=core_node1 # 确保各个节点没有重复的 coreNodeName
```

然后，随机挑选一个主机，通过图 9-4 中左侧下方的 Core Selector 选中 product 这个 core，并打开 File 选项，这时就可以看到配置文件列表了，它们都是通过 ZooKeeper 同步而来的。图 9-5 显示了一个节点上的 schema 配置，和我们上传之前的定义一样。

当然，目前还没有索引任何数据，这可以通过打开左下方的 Query 选项来验证。该选项打开的是一个查询设置的界面，执行默认的全量查询，从图 9-6 中可以看出返回的内容提示没有任何结果。

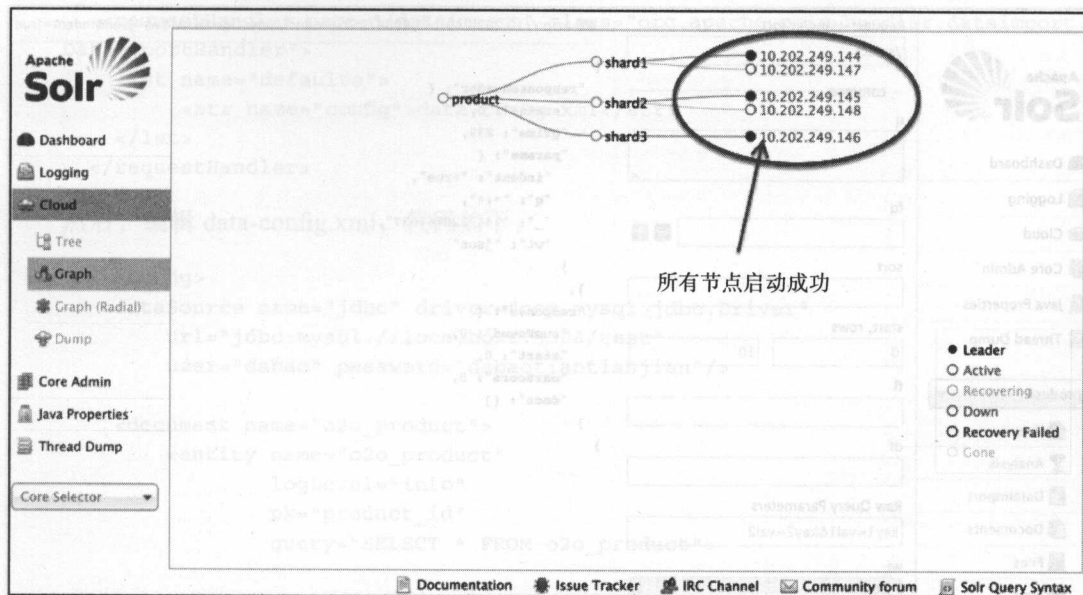


图 9-4 总共 5 台 Solr 服务启动成功

3. 看到之前的配置

1. 选择 Product 的 Core

2. 选择 Files

```

<!-- fields for product -->
<field name="product_id" type="long" indexed="true" stored="false" />
<field name="product_name" type="chinese_ik" indexed="true" stored="true" />
<field name="category_name" type="chinese_ik" indexed="true" stored="true" />
<field name="product_pic" type="string" indexed="false" stored="true" />
<field name="product_price" type="double" indexed="false" stored="true" />
<field name="product_stock" type="int" indexed="false" stored="true" />
<field name="product_desc" type="string" indexed="false" stored="true" />
<field name="seller_name" type="string" indexed="true" stored="true" />

<uniqueKey>product_id</uniqueKey>

<!-- DEPRECATED: The defaultSearchField is consulted by various query parsers when
 parsing a query string that isn't explicit about the field. Machine (non-user)
 generated queries are best made explicit, or they can use the "df" request parameter
 which takes precedence over this.
 Note: Un-commenting defaultSearchField will be insufficient if your request handler
 in solrconfig.xml defines "df", which takes precedence. That would need to be removed.
<defaultSearchField>text</defaultSearchField> -->
<defaultSearchField>product_name</defaultSearchField>

<!-- DEPRECATED: The defaultOperator (AND|OR) is consulted by various query parsers
 when parsing a query string to determine if a clause of the query should be marked as
 required or optional, assuming the clause isn't already marked by some operator.
 The default is OR, which is generally assumed so it is not a good idea to change it
 globally here. The "q.op" request parameter takes precedence over this.-->
<solrQueryParser defaultOperator="AND"/>

<!-- copyField commands copy one field to another at the time a document
 is added to the index. It's used either to index the same field differently,
 or to add multiple fields to the same field for easier/faster searching. -->

```

图 9-5 从 ZooKeeper 同步过来的配置文件

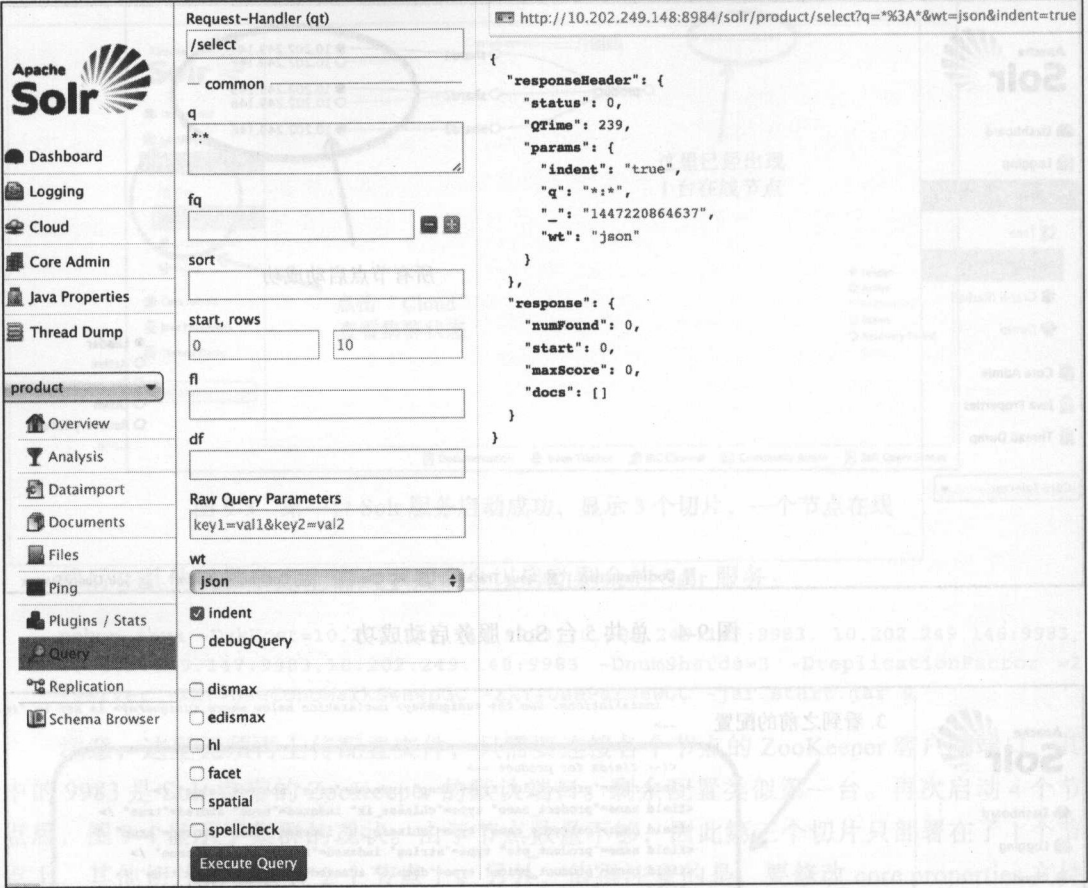


图 9-6 默认查询返回所有的结果，目前没有任何索引数据

9.3.3 DIH 配置

对于导入源数据的步骤，目前的方案是采用 DIH（Data Import Handler）直接将数据从关系型数据库 MySQL 里导入到 Solr 集群。在开始配置 DIH 之前，首先要确保 DIH 的 jar 包和 MySQL 的 JDBC 连接 jar 包都已经放在 \solr-webapp\webapp\WEB-INF\lib 目录下了，否则运行数据导入时会抛出驱动加载的异常。例如，4.8.1 版本 Solr DIH 的 jar 包是 solr-dataimporthandler-4.8.1.jar，可以在 dist 目录中找到。而在 MySQL 的官网（<http://dev.mysql.com/downloads/connector/j/>）可以找到连接器，现在的版本是 mysql-connector-java-5.1.37-bin.jar。

停止 Solr Cloud 的集群，选择第一个节点打开 solrconfig.xml，指定 DIH 的配置文件 data-config.xml，代码如下：

图 9-2 从 ZooKeeper 同步过来的配置数据



```

<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.
DataImportHandler">
  <lst name="defaults">
    <str name="config">data-config.xml</str>
  </lst>
</requestHandler>

```

然后，编辑 data-config.xml，代码如下：

```

<dataConfig>
  <dataSource name="jdbc" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/test"
    user="dabao" password="dabaotiantianjian"/>

  <document name="o2o_product">
    <entity name="o2o_product"
      logLevel="info"
      pk="product_id"
      query="SELECT * FROM o2o_product">

      <field column="product_id"          name="product_id"/>
      <field column="product_name"        name="product_name"/>
      <field column="category_name"       name="category_name"/>
      <field column="product_pic"         name="product_pic"/>
      <field column="product_price"       name="product_price"/>
      <field column="product_stock"       name="product_stock"/>
      <field column="product_desc"        name="product_desc"/>
      <field column="seller_name"         name="seller_name"/>

    </entity>
  </document>
</dataConfig>

```

其中 dataSource 指定了 JDBC 的连接参数，包括主机 IP、MySQL 的端口号、用户名和访问密码等。而 document 中，pk 是数据库主键，query 是查询的 SQL 语句，field 用于将数据库字段和 Solr 的字段进行匹配。本样例中数据库字段的名称和 Solr 字段的名称恰好一致，所以看上去有点怪。实际应用中对于解决名称不匹配的遗留问题非常有利。配置完毕，再次按照上一节的步骤，启动整个 Solr Cloud 集群。之后，按照图 9-7 所示的步骤开始进行数据的导入，默认的是全量导入（full-import）。选项 Clean 表示清除之前的索引，要谨慎使用。而 Commit 表示将索引变化从内存中持久化到磁盘，Verbose 和 Debug 可以提供更多信息用于调试，Optimize 用于索引结构的优化。图 9-8 表明我们已经将 MySQL 中的 5 条测试记录全部导出，并在 Solr 中建立了索引，随后 Solr Cloud 的机制还会帮助将索引数据传播到其他相应的分布式节点上。图 9-9 展示了查询所有结果后的内容，一共返回了 5 条记录。在全量索引的时候，还可以通过 MySQL 的 Limit 和 Offset 进行批量的导出，以避免过大的资源消耗。此外，还可以配置增量的 DIH 配置，每次只更新有变化的数据。

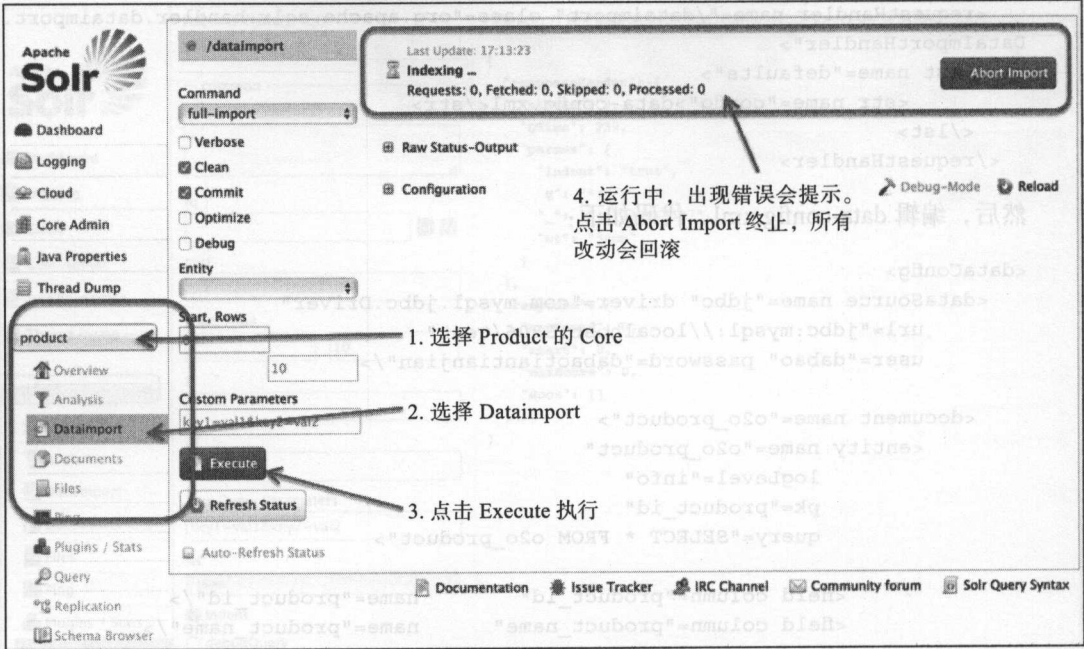


图 9-7 数据导入和索引建立过程中，如果出现错误会予以提示

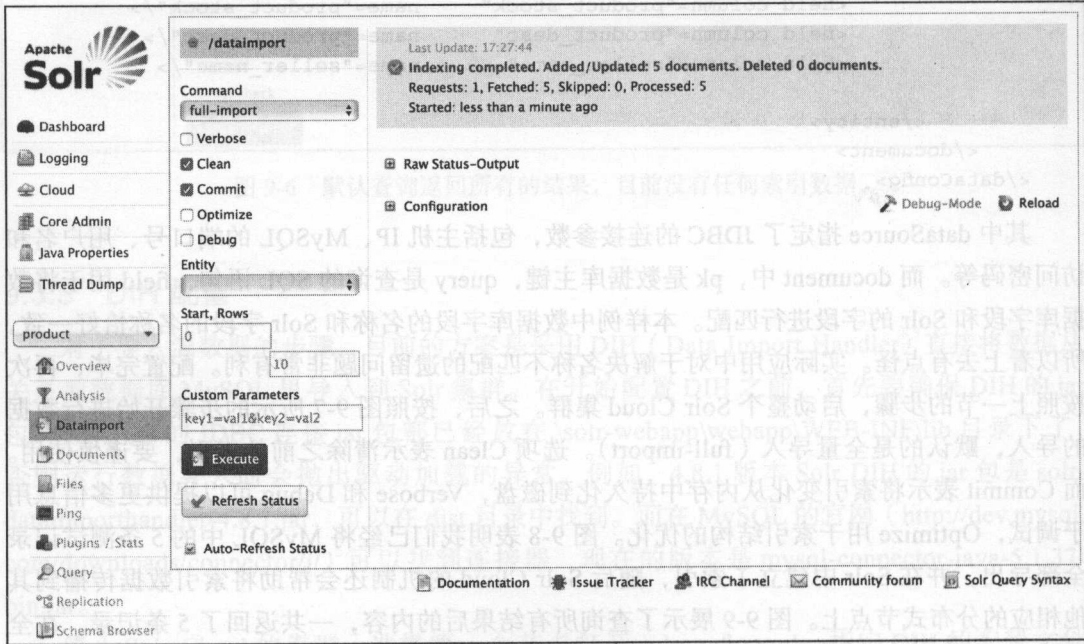


图 9-8 数据导入和索引建立完毕，本次共处理了 5 条记录

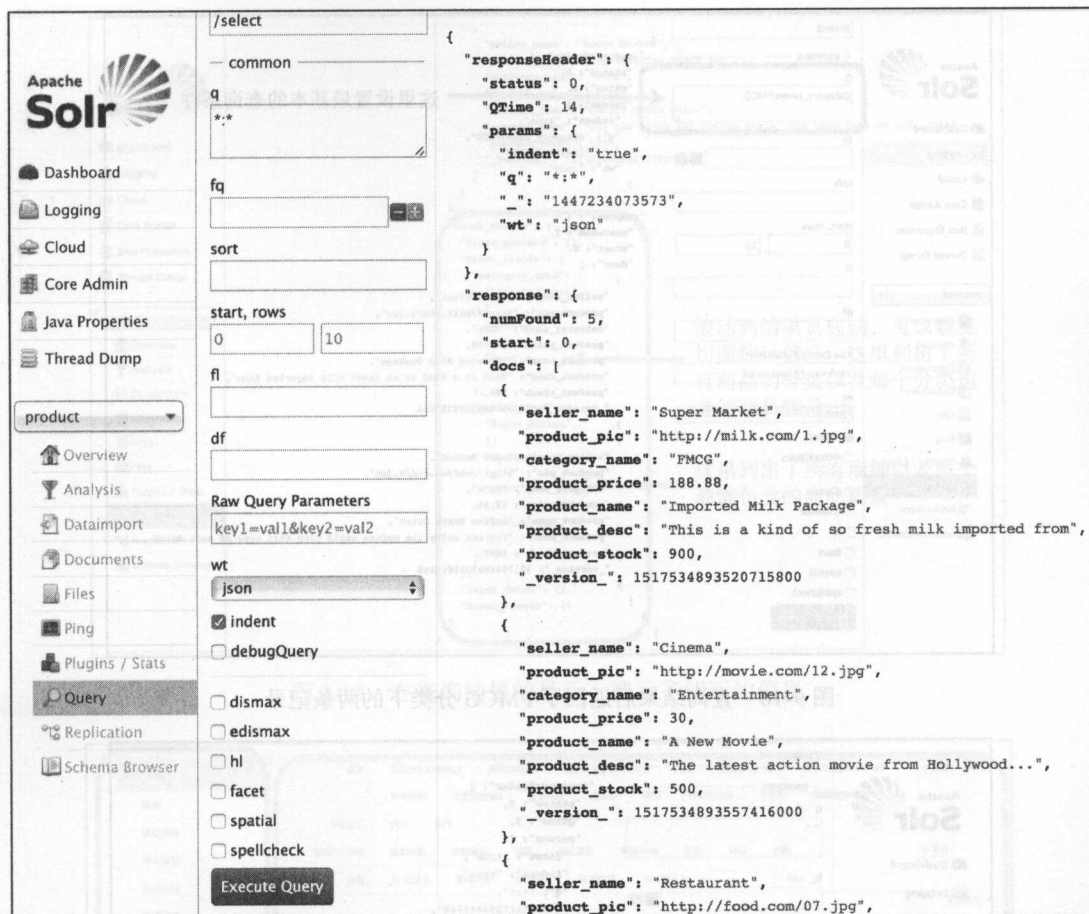


图 9-9 查询全部结果后返回了 5 条记录，默认是 JSON 格式

若要指定查询的条件，还可以设置 `q` 参数。例如，`category_name:FMCG` 表示只搜索类别为快速消费品（Fast Moving Consumer Goods）的商品。这样就只返回了两个相关的结果（如图 9-10 所示）。

最后，图 9-11 展示了切面（Facet）的用法，在 Raw Query Parameters 里输入如下代码：

```
&facet=true&facet.field=category_name&facet.field=seller_name
```

其中，`facet=true` 表示采用切面选项，`facet.field=category_name` 和 `facet.field=seller_name` 表示分别按照商品的分类和商家这两个维度来计算切面信息。这样在查询出来的搜索结果中，还会附上切面统计的信息，如图 9-12 所示。这对于搜索结果中的细分类和导购属性都是必不可少的，图 9-13 展示了前端的样例。





图 9-10 查询结果后返回了 FMCG 分类下的两条记录



图 9-11 设置切面 Facet 的查询

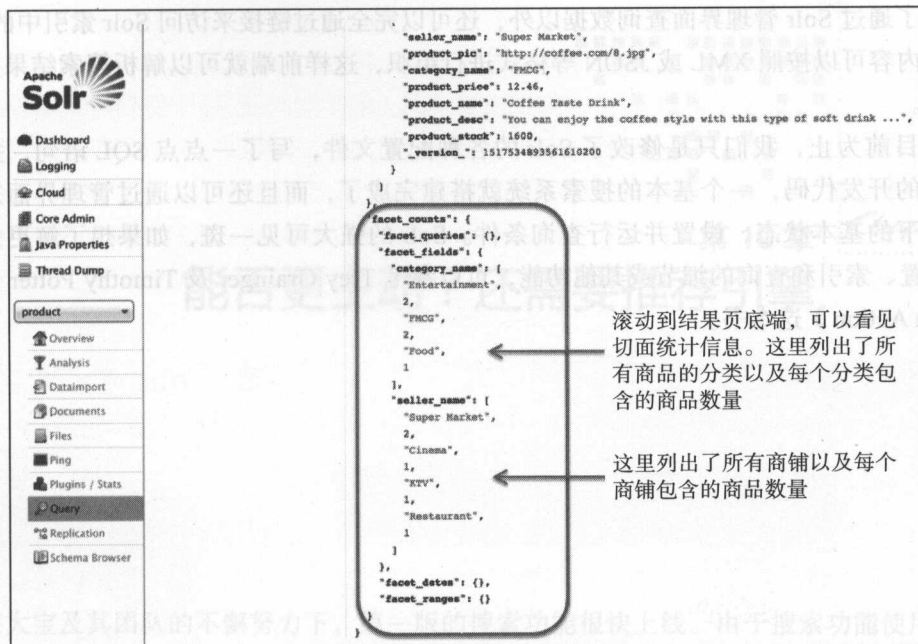


图 9-12 在查询结果的最后，展示了切面的信息



图 9-13 前端展示的筛选项往往都是采用切面技术来完成的

除了通过 Solr 管理界面查询数据以外，还可以完全通过链接来访问 Solr 索引中的数据，返回的内容可以按照 XML 或 JSON 等格式进行组织，这样前端就可以解析搜索结果并予以展示了。

到目前为止，我们只是修改了 Solr 的各种配置文件，写了一点点 SQL 语句，并没有写任何的开发代码，一个基本的搜索系统就搭建完成了，而且还可以通过管理界面来查看集群当下的基本状态，设置并运行查询条件。Solr 的强大可见一斑，如果想了解更多关于 Solr 配置、索引和查询的细节或其他功能，可以参考 Trey Grainger 及 Timothy Potter 合著的《Solr in Action》这本书。

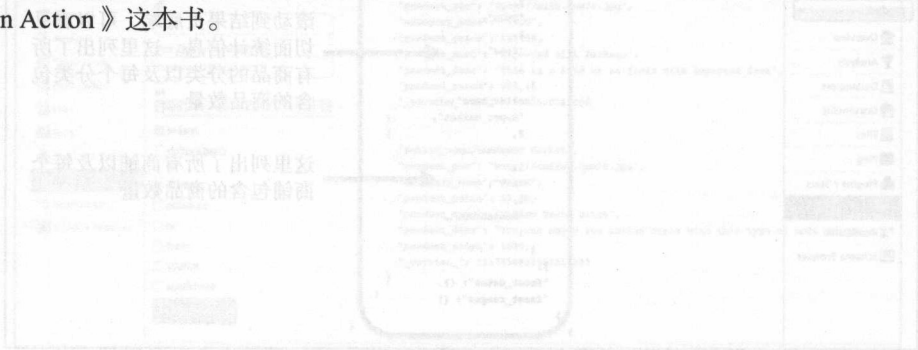


图 11-2 显示的面板下视图，这里就是查询条件 UI 界面

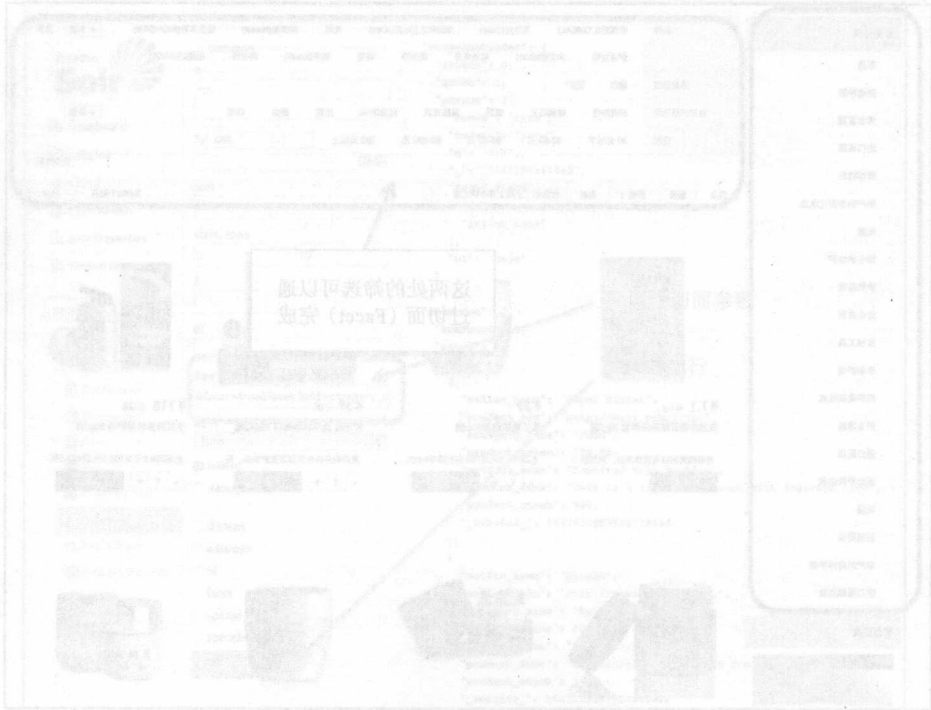


图 11-3 显示的面板上视图，这里就是查看集群基本状态 UI 界面



## 能否更主动？还需要推荐引擎

在大宝及其团队的不懈努力下，第一版的搜索功能很快上线。由于搜索功能使用便捷，用户的反响非常热烈，流量的转化率也有了明显的提升。业绩上了一个台阶，公司的合伙人稍稍松了口气。不过，大宝清晰地记得小丽曾说过，排在前两位的都是和技术相关的痛点，他很想知道第二个问题具体是什么。于是，他主动找到了小丽。

### 10.1 业务需求

“嗨，小丽，忙什么呢？有空聊一下吗？”

“大宝，你好。这不是春节快到了嘛，我们运营部正在积极酝酿活动的方案，还有些市场推广的策划也在进行，所以最近超级忙。你找我有事吗？”

“其实我很想知道，你上次说的用户调研中，抱怨比较多的第二个问题是什么。”

“嗯，这个事情的优先级也很高，正好我们可以坐下来详细地谈一谈。首先，非常感谢你们团队的快速响应，搜索功能上线后我们用户的活跃度和订单量都明显增长了。至于第二个问题，无论是从用户的角度，还是从我们公司的角度来看，该需求都是很有必要的，从本质上来说我认为就是如何进一步增加销售。”

“双方都有需求？什么叫‘增加’我们的销售？”

“对，简单地讲就是增加推荐栏位。这种技术在各大互联网站点已经得到普遍使用，对于用户来说，他们即使没有输入搜索条件，系统也会提出一些建议，帮助他们发现一些可能感兴趣的<sup>①</sup>商品。而对于公司来说，这也是增加跨品类销售和向上促销<sup>②</sup>的绝好机会。在这方

① 来自英文 upsell，就是让顾客买更贵或更多的商品来达到增加销售额的目的。

面，最经典的案例应该是美国的亚马逊电子商务网站。亚马逊是全球最大的 B2C 电商网站之一，在公司创立之初，最为出名的就是其丰富的图书品类，以及相应的推荐技术。该公司的推荐系统会根据用户的历史行为，推荐更符合他 / 她预期的商品。”说到这里，小丽又打开了她的电脑，“比如，从某位 IT 宅男的亚马逊主页可以看出（如图 10-1 所示），亚马逊根据其个人的浏览记录，产生了一个综合的推荐栏位，且此栏位占据了首页不少的资源位。从推荐的内容来看，此用户对大数据和互联网技术很感兴趣，同时最近也非常关注智能小家电。事实上，我们公司也很需要这项技术，如此一来不用被动地等待用户输入关键词，也能产生潜在的销售。”



图 10-1 某位用户亚马逊主页的推荐内容

“原来是这样啊,这个课题我之前也有所了解。”这番谈话马上就让大宝想起了小明的指导课程,他接着说道,“通常,我们认为传统的搜索主要是利用全体用户的整体行为,而推荐则更侧重于挖掘个人的喜好。另外,搜索会要求用户输入具体的关键词,而推荐往往没有要求明确的查询条件,主要是通过用户之前的历史行为数据,以及当下的应用场景,然后根据算法分析进行大致的推测。所以推荐相对于搜索而言,会更模糊一些,也更具挑战性。具体的方案我还要和团队一起商量才能制定。”

“不愧是我们的技术大牛,好了,这个需求交给你我就放心了。我接着去忙那个市场策划案了。”

## 10.2 产品设计和技术选型

回到自己的座位,大宝开始陷入沉思。在第5章的推荐部分,小明介绍了推荐系统的主要概念、核心要素和流程框架,也推荐了 Mahout 这样的开源项目。不过,大宝感觉这些还是比较偏向理论和技术层面,离实际的商业需求还有相当的距离,一时间有些迷茫,没了方向。于是他再次将业界经验丰富的表哥约到了茶室。

“大宝,你的情况我基本了解了。推荐系统的设计和开发,要比搜索引擎更高端一些。如果刚开始就设计得过于精细,可能会导致开发周期过长,技术难点引入的风险也会增大。这样吧,今天先讨论下最基础的版本。”

“好的,完全没问题,我洗耳恭听。”

“前面你提到的一点很好,就是要注意应用的场景,我也同意这个观点。考虑到你们是偏向于电子商务的业态,建议从在线购物的角度入手,依次分析下可能的推荐栏位。”

在分析每个具体的栏位之前,先介绍一下商品购买模式中常用的2个基本概念:相似品(Similar Item)和相关品(Related Items)。相似品是指相互之间有很高的替代度的商品,例如不同口味的薯片、不同品牌的牛奶,或者是品牌相同但是型号不同的手机。而相关品一般都是跨品类的,虽然它们属于不同的商品类目,但是从消费行为来看存在很高的关联性,例如奶瓶和尿布、电脑整机和配件、衬衣和领带这样的关系。

对于第5章中介绍的推荐分类,可以从不同的维度来理解。这里按照针对用户还是商品来进行基本的拆解。

针对用户,进一步划分为群体和个人。

□ 群体:根据全站用户的行为来推荐,最经典的栏位就是“热销排行榜”这种。由于没有过高的技术难度,本文就不做深入探讨。

□ 个人:根据某位用户个人的历史行为来推荐,这种一般应用在“个性化主页”、“用户中心”等栏位,图10-1的亚马逊主页就是一个例子。最基础的是提供用户之前浏览过,但是没有购买的商品。对于已经购买过的商品,需要考虑的是商品类型和周期性。如果是快消品,那么在周期过后可以考虑推荐相似品类。如果不是,那么很



可能要推荐相关品类。如果再进一步拓展，则不必仅限于用户浏览或买过的商品了，还可以通过其他可获取的个人信息来设计，包括他/她的职业背景、兴趣爱好、喜好品牌、消费能力等。“用户中心”可能强调更多的是之前已购商品的重复购买，而“个性化主页”则重点强调的是对于尚未购买商品的再次推广。

针对商品，可以划分相似商品和相关商品两个部分。

□ 相似商品：如前所述，商品的相互替代性高，一般应用在购买决策之前，便于用户进行商品的比较。常见的商品详情页“看了此商品的用户还看了”、“看了此商品的用户最终买了”等栏位均属于此类，如图 10-2 所示。

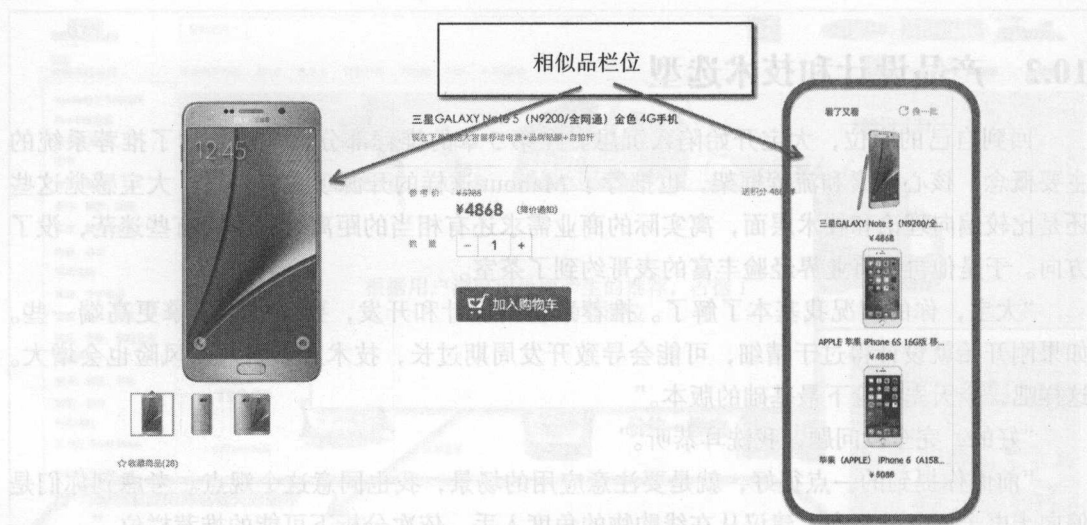


图 10-2 相似品栏位示意

□ 相关商品：商品之间存在着关联性，而不是替代性。因此，多用于用户购买后进行跨品类的关联销售，以扩大销售额。通常商品详情页的“购买此商品的用户还买了”、“购买此商品的用户还看了”等栏位属于此类，如图 10-3 所示。

除了根据用户或商品来划分外，还有一个综合类，这是比较复杂的场景，可能要同时考虑用户和商品的信息。例如当用户的购物车里已经添加了若干商品，进入结算页时，我们也可以提供建议，让其购买更多可能感兴趣的物品，或是进行免邮凑单。这时候，既可以考虑和购物车中的商品相似、相关的品项，也可以考虑用户个人的喜好。

此外，之前还介绍了一个非常重要的推荐框架：协同过滤（Collaborative Filtering）。协同过滤是基于最直观的“口口相传”，利用已有用户群过去的行为或意见，预测当前用户最有可能喜欢哪些东西。根据推荐依据和传播的路径，又可以进一步细分为基于用户的过滤和基于物品的过滤。因此，对于上述不同的场景都可以采用协同过滤的方法，进一步扩大推荐的范围，给用户带来更多的新鲜感。“看了此商品的用户还看了”、“看了此商品的用户最终

买了”、“购买此商品的用户还买了”和“购买此商品的用户还看了”等利用用户访问商品的行为特征衡量商品之间相似程度的,都可以归为基于物品的协同过滤。由于这些栏位的名称已经阐明了推荐的逻辑,所以它们很容易被人们所理解。而对于针对用户的推荐,如果需要应用基于用户的协同过滤,则需要利用用户访问商品的行为特征对人群进行聚类或分组,形成“近邻”。然后对于当前用户没有访问过的物品,利用其同组近邻的访问记录来预测和推荐。由于用户分组这类信息一般不会在前端展示给顾客<sup>①</sup>,因此相对于基于物品过滤的算法更为隐蔽一些,也不容易为人所知。同时,一般情况下协同过滤不会特意区分相似品和相关品,完全是基于用户的行为来预测和推荐的。因此,推荐结果取决于统计的行为分类,以及时间窗口,需要结合具体的场景来对待。最后,比较下协同过滤和数据库里的关联规则挖掘。从模型的角度而言两者是比较接近的,不过也有不同点。严格说来,关联规则面向的是成交,目的非常明确。而协同过滤面向的是用户偏好,不一定是购买,比较模糊。所以,协同过滤的约束条件没有关联规则那么强,但是同时也更为灵活,可以融入更多的商业规则。

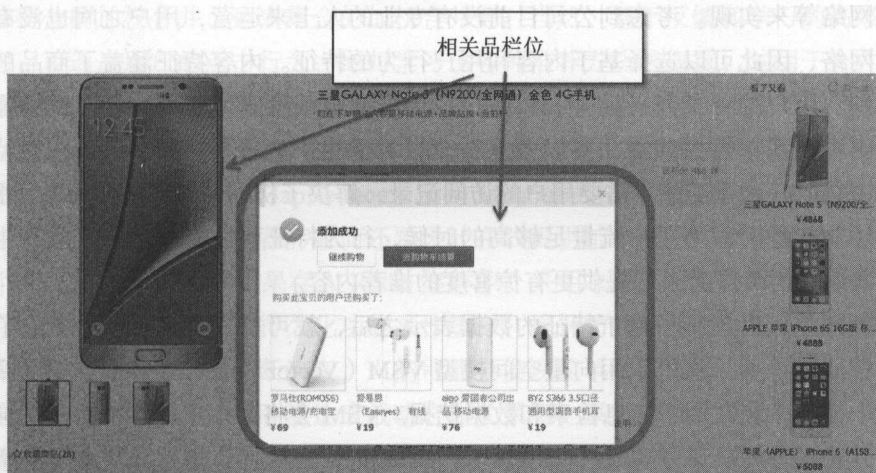


图 10-3 相关品栏位示意

综上所述,可以将常见的推荐栏位进行如表 10-1 所示的划分。需要注意的是,这里的划分只是一个基本的框架,在实际运用中可能还要结合具体的业务需求进行定制和融合。

理解了在不同场景下推荐的大致逻辑和协同过滤的方法后,我们来设计各个模块和整体的架构。这里主要的系统模块包括:相似度计算、协同过滤、结果缓存。前两者是离线计算的部分,而最后一个模块主要是服务于在线访问的。相似度计算主要涉及两点要素:数据特征选取和模型建立。

<sup>①</sup> 也有例外,例如按照地理位置分组的“附近的人”。

表 10-1 常用推荐栏位的划分

	基于内容的特征	基于访问行为的特征
针对用户	用户中心、个性化主页 (侧重相似品, 相关品需要人工运营规则)	个性化主页、用户中心 (侧重相关品)
针对商品	和本商品相似的其他商品 (用户访问流量缺乏的情况下, 而且侧重相似品, 相关品需要人工运营规则)	看了此商品的用户还看了 看了此商品的用户最终买了 购买此商品的用户还买了 购买此商品的用户还看了 (用户访问流量充足的情况下, 可能同时包含相似品和相关品)
综合	购物车中的关联销售、免邮凑单 (侧重相似品, 相关品需要人工运营规则)	购物车中的关联销售、免邮凑单 (侧重相关品)

- ❑ 数据特征选取：确定如何将现实中的对象转换为计算机可以理解的数据。第 6 章一直在探讨的水果案例就多次提及了如何选择特征以用于水果分类。从第 5 章的介绍中可以得知，在推荐系统中计算相似度时可以基于内容、用户行为、专业知识和社交网络等来实现。考虑到公司目前没有专业的人士来运营，用户之间也没有形成社交网络，因此可以选择基于内容和用户行为的特征。内容特征涵盖了商品的标题和类别、用户的背景和兴趣等。而用户行为则涵盖了顾客浏览或购买了哪些商品。基于内容和用户行为的特征可以相互补充。对于用户流量不够的情形，内容特征是首选，它的好处在于并不需要用户的访问记录，解决了机器学习系统中常常面临的“冷启动”问题。而当用户流量足够高的时候，行为特征可以挖掘从文字本身中无法发现的潜在语义，为人们提供更有惊喜度的推荐内容。
- ❑ 模型建立：对象有了基于特征的数据表示之后，就可以通过模型来分析它们了。对于相似度计算，这里采用向量空间模型 VSM (Vector Space Model)。第 5 章有介绍这个模型，它常用于信息检索和数据挖掘。VSM 会将某个文档转换为一个向量，统计其中的单词，若去重后有 n 个不同 (Unique) 的单词，那么该文档的向量就是 n 个维度。这里每个维度都可以选择其对应单词的 *tf-idf* 值。其中，*tf* 表示词频 (term frequency)，就是一个词在某篇文章中出现的次数，*tf* 值越高，表示该词对于文档而言越重要。而 *idf* 表示逆文档频率 (inverse document frequency)，其假设是，某个词在文档集合中出现在越多的文档中，那么其重要性就越低，反之则越高。这种表示忽略了单词在文章中出现的顺序，大大简化了很多模型的计算复杂度，同时保证了相当的准确性，适合较短的文本，例如商品的标题。最后，相似性问题就转化为了计算两篇文档向量之间的余弦距离。该值正好是一个介于 0 到 1 之间的数，越接近于 1 则越相似。从实现的角度而言，可以选择开源的 R 语言来计算向量之间的余弦距离。



如果确定使用用户行为,那么协同过滤就是必不可少的框架,可以通过 Apache 的 Mahout 来实现它。Mahout 对于基于用户的推荐和基于物品的推荐都有实现,它会将用户对物品的偏好形成一个二维矩阵,并以一个用户对所有物品的偏好作为向量来计算用户之间的相似度,或者将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度。Mahout 关于相似度计算的实现都是基于向量距离的,因此很容易实现集成 VSM 模型和余弦距离的计算。随着客户的访问量日益增大,访问数据的规模也会不断地膨胀。好在 Mahout 从设计开始就旨在建立可扩展的机器学习软件包,因此某些部分的实现(例如矩阵运算)都是直接基于 Hadoop 的 MapReduce 计算框架<sup>①</sup>的,这就使得其具有了处理大数据的能力,这也是 Mahout 最大的优势所在。

如果同时使用了基于内容和基于用户行为的推荐,或者是同时使用了基于用户和物品的推荐,则要考虑推荐的混合模型。如果使用微观混合,开发者需要将不同的特征混合起来,这就会涉及修改 Mahout 的源码了,工作量相对较大。宏观混合则不用关心特征的合并,它可将基于内容的推荐和基于行为的推荐结果有机地合并起来。合并的策略也有多种选择,例如得分的加权合并、列表集合的加权合并等。当然,基于反馈和学习的高级策略同样可以考虑,可通过用户的点击行为,回归学习出各种推荐方法的权重,并用于合并的参数调优。只是在第一版中建议尽量简洁,暂不考虑这个复杂的功能。

有了数据的特征定义、相似度模型和 Mahout 的计算,我们就可以完成数据挖掘的部分。但是,无论是 Hadoop 的 MapReduce,还是 Mahout 的协同过滤模型,都是批处理方式,因此没法进行实时性很强的推荐。故而还要设计在线提供结果的接口。常用的做法之一是将针对每个商品或顾客的推荐结果存入 Redis 集群中。采用 Redis 的主要原因是既可以利用其长期保存结果,也能利用其非持久化功能来作缓存,提升访问的速度。当然,随着市场规模日渐扩大,以及业务逻辑的不断演变,人们可能会发现简单的键-值(Key-Value)查询模式无法满足需求。例如,和某品牌商达成战略合作关系后,系统需要在某些地区加大该品牌的推广力度,将其商品在推荐栏位里适度提前。这里在查询推荐分析结果的时候,至少要额外加入地域和品牌两个因素,如果仍然使用 Redis 实现这项需求,那么只能设计非常复杂的键结构,而且还会导致数据的冗余,不利于长期的性能调优和整体维护。此时可以考虑利用 Solr 这类的搜索引擎,我们只需要将推荐挖掘的结果和地域、品牌等其他商品信息一并建入搜索索引,就可以完成多条件的复杂查询。

依照这些模块搭建的推荐系统的大体架构如图 10-4 所示。如果需要特别强调相关品的推荐,可以加入基于 R 语言的关联规则挖掘。如果根据应用场景,需要不同的合并策略,可以在 Redis 集群中加入参数设置,存储不同的结果集合,甚至是修改 Solr 集群中的索引数据。

① Mahout 最新的动态是支持 Spark,原理和基于 MapReduce 的 Mahout 类似。

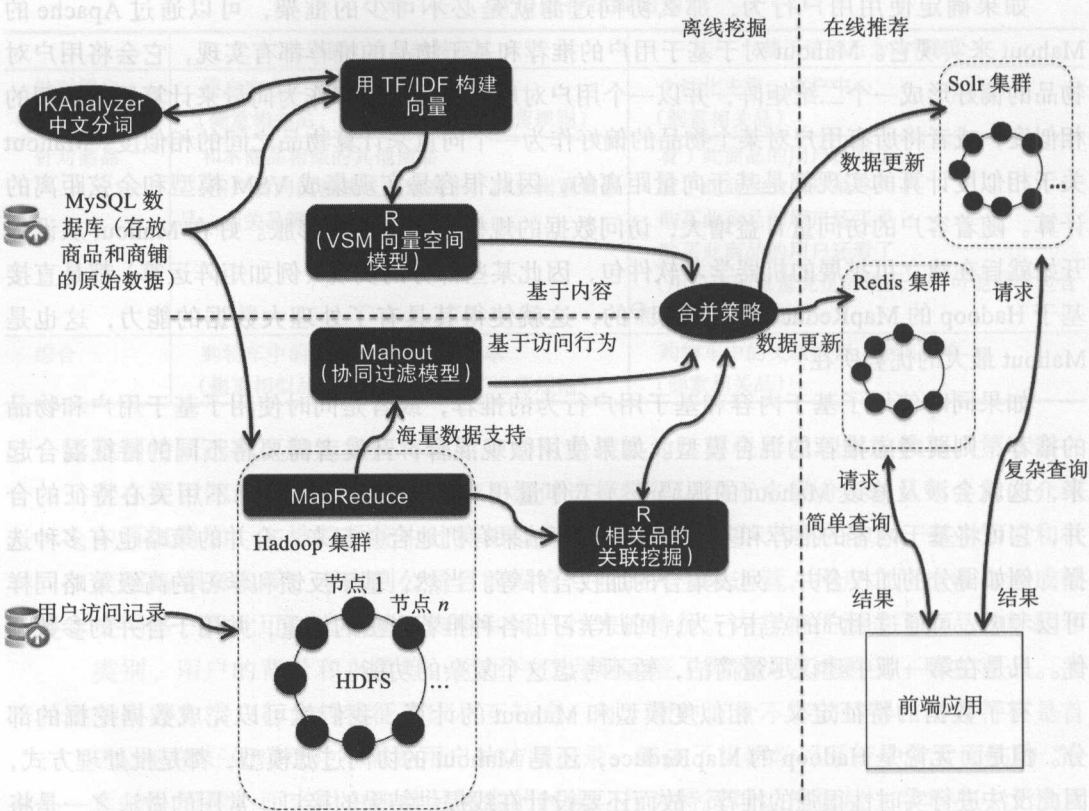


图 10-4 VSM 模型、Mahout 协同过滤模型和 Redis 集群搭建的推荐系统架构

## 10.3 实现方案

下面分别从相似度、协同过滤、缓存模块等方面，探讨推荐引擎的基本实现方式。

### 10.3.1 基于内容特征的衡量

本案例将针对电商 O2O 网站的商品推荐进行分析，其基于内容的主要特征选择如表 10-2 所示。

表 10-2 商品的常见内容特征选择

名称	含义
商品名称	商品名的全称，例如“日式精致美甲美睫套餐”。需要中文切词的支持
商品属性	从可读性和简洁性来考虑，名称有时无法描述商品所有的内涵。同时，名称也没有办法体现哪些方面对于商品而言是更为重要的。因此，经常要考虑运营给商品设置的属性。由于属性结合了人类的知识，因此通常要给予其更高的权重。例如某款牛奶还有“低脂”、“高钙”、“助消化”、“进口”、“澳大利亚”等这样的属性。为了避免歧义，一般不进行中文切词处理

(续)

名称	含义
商品分类	由于商品的名称和属性是基于词包 (Bag of Word) 的方式来处理的, 并未考虑其语义, 因此引入分类这维特征是一个很好的补充。例如, 词包的处理可能会认为“巧克力口味的牛奶”和“牛奶口味的巧克力”之间的相似度很高, 但是“巧克力口味的牛奶”和“低脂高钙牛奶”的相似度并不高。但是如果加入分类的概念, 系统就能理解“巧克力口味的牛奶”和“低脂高钙牛奶”都是属于牛奶这个分类, 而“巧克力口味的牛奶”和“牛奶口味的巧克力”分属于不同的食品分类, 从而在相似度衡量上做出新的判断
商品的卖家	这里有个假设, 就是同一个专卖店卖出的商品具有某种程度的相似或相关性。例如母婴店里不同品牌的奶瓶就有相似性, 奶瓶和奶粉、尿布等则具有相关性。一般不进行中文切词处理。 当然, 这些假设对于综合卖场并不适用。另外, 从人和物的对应关系思考, 我们也可以认为这个特征属于用户行为, 不过相对于顾客的浏览和购买而言, 卖家的销售行为基本是固定不变的, 因此可以简化为内容特征来处理。从中也可以看出, 行为特征和内容特征其实也是可以相互转换的

除了商品, 另外一个角色就是用户, 其主要内容特征如表 10-3 所示。由于这里会涉及人, 因此用户基于内容的特征和访问行为特征在概念上很容易产生混淆。所以这里强调的是除了商品浏览和购买的具体行为之外的特征。例如, 表 10-3 中提到的消费属性, 它只关心消费的金额、频次和周期等, 并不考虑具体的购买物品及基于此的协同过滤。

表 10-3 用户的常见内容特征选择

名称	含义
用户社会属性	包括性别、年龄、职业、爱好等, 这里假设有类似社会属性的用户之间更相似。不过, 对于互联网而言, 这类信息比较难以获得准确的资料
用户消费属性	购买的金额、频次和周期, 这里假设有相似消费习惯的用户更相似
用户的地理属性	需要用户提供具体的位置, 可能是工作、娱乐和生活场所。这里假设在邻近商圈或住宅区出现的用户更相似
用户的设备属性	用户访问网站时使用的设备, 例如 PC、何种品牌的手机、操作系统等。这里假设使用类似设备的用户更相似

对每个商品或用户的内容特征进行必要的中文切词后, 统计每个关键词的词频  $f$  和逆文档频率  $idf^{\ominus}$ , 就可以为每个商品生成对应的向量。向量之间的夹角余弦计算非常简单, 可以自行编码实现, 也可以利用 R 语言<sup>②</sup>来解决。这里输入 3 个向量  $x$ 、 $y$  和  $z$  作为测试, 代码如下:

```
> x <- c(0.25, 0.28, 0.87, 0.11, 0.62, 0.96, 0.28, 0.08, 0.67, 0.31, 0.75,
0.95, 0.85, 0.68, 0.27, 0.22, 0.78, 0.53)
> y <- c(0.58, 0.84, 0.65, 0.16, 0.32, 0.03, 0.86, 0.11, 0.76, 0.96, 0.41,
0.19, 0.68, 0.28, 0.25, 0.24, 0.26, 0.03)
> z <- c(0.26, 0.38, 0.86, 0.23, 0.60, 0.88, 0.45, 0.12, 0.65, 0.25, 0.48,
0.93, 0.81, 0.62, 0.24, 0.28, 0.85, 0.52)
> x
```

<sup>①</sup> 如果商品的数量规模达到一定程度, 也可以使用 MapReduce 进行处理。

<sup>②</sup> 本文用的 R 版本是 3.2.2。



```
[1] 0.25 0.28 0.87 0.11 0.62 0.96 0.28 0.08 0.67 0.31 0.75 0.95 0.85 0.68
0.27 0.22 0.78 0.53
> y
[1] 0.58 0.84 0.65 0.16 0.32 0.03 0.86 0.11 0.76 0.96 0.41 0.19 0.68 0.28
0.25 0.24 0.26 0.03
> z
[1] 0.26 0.38 0.86 0.23 0.60 0.88 0.45 0.12 0.65 0.25 0.48 0.93 0.81 0.62
0.24 0.28 0.85 0.52
```

向量 x 和 y 之间的夹角余弦计算如下：

```
> cosxy=sum(x*y)/sqrt(sum(x^2)*sum(y^2))
> cosxy
[1] 0.6891425
```

同样，x 和 z、y 和 z 的夹角余弦计算为：

```
> cosxz=sum(x*z)/sqrt(sum(x^2)*sum(z^2))
> cosxz
[1] 0.988296
> cosyx=sum(y*z)/sqrt(sum(y^2)*sum(z^2))
> cosyx
[1] 0.7183958
```

从上面的计算结果可以看出，向量 x 和 z 之间的夹角更小，相似度更高。而向量 y 和 x、z 的相似度则比较小。

另外，如果数据量很大，单机难以应付，则可以考虑将 R 和 Hadoop 结合使用。在沒有 Hadoop 家族之前，人们对于大数据的处理往往是获取样本、假设检验、做回归等方式。而 Hadoop 的强大之处在于对大数据的处理，让 TB、PB 级别的数据量计算成为了可能。将 Hadoop 的规模处理能力和 R 语言的数据分析能力相结合，正好可以取长补短。常见的结合方法有三种。

❑ RHadoop：这是一款结合 Hadoop 和 R 语言的产品，由 Revolution Analytics 公司开发，并将代码开源到了 Github 社区上。RHadoop 包含三个 R 包，即 rmr、rhdfs 和 rhbase，分别对应于 Hadoop 系统架构中的 MapReduce、HDFS 和 HBase 三个部分，使其可以在客户端和 MapReduce 进行无缝集成。

❑ Rhipe：和 RHadoop 类似，也是一个开源项目。其在客户端将 MapReduce 和 R 紧密集成，可以直接通过 R 的环境进行访问。

❑ R+Streaming：在这个方式中，使用 MapReduce 执行 map 和 reduce 的 R 脚本。可以对 MapReduce 函数进行比较方便的调用，但是从另一个角度而言，很难从 R 中直接调用。

对于 R 语言的细节，可以参考 Robert I. Kabacoff 所著的《R 语言实战》。

对于 R 和 Hadoop 的结合，可以参考 Alex Holmes 所著的《Hadoop 硬实战》。

### 10.3.2 基于行为特征的衡量

同样，对于用户行为，首先要考虑哪些行为特征可以纳入考量。表 10-4 列出了常用的部分<sup>①</sup>。

表 10-4 常见的用户访问行为特征

名称	含义
正向：表达用户的喜欢程度	
浏览	用户点击单个商品后浏览其详情页。由于通常情形下我们无法跟踪用户的眼睛关注的是商品列表中的哪一款，因此都是以他/她实际的点击和打开为准。如果打开某款商品的详情页，就假设用户对其感兴趣
收藏	如果用户将某款商品加入其收藏夹，则假设用户对其感兴趣
加入购物车	如果用户将某款商品加入其购物车或购物篮，则假设用户对其感兴趣
付款购买	如果用户最终付款购买了某款商品，那么这个兴趣是相对明显和确定的
好评	在收到商品甚至是使用一段时间后，用户留下了肯定的评语，这也能体现其对该商品的认可
负向：表达用户厌恶的程度	
差评	在收到商品或使用一段时间，甚至是退换货之后，用户留下了抱怨的评语，这说明商品服务的某些方面未能达到用户的期望，很可能会阻止用户再次购买类似的商品，或者阻止用户在同一家店铺再次消费
退换货	如果用户最终选择了退换商品，这肯定是一个非常负面的信号，有的时候可能比留下差评还要糟糕

对于这些特征，或者说是不同类型的行为，应该赋予不同的权重。比如，浏览、收藏、加入购物车、付款购买和留下好评，代表用户对于商品喜好程度由浅到深，自然在计算时的权重也需要从低到高。此外，目前人们考虑的比较多的是正向的特征，很少考虑负向特征，但是对于越来越讲究用户体验的互联网而言，负面的信息同样重要。甚至有些算法已经开始设计针对负向特征的推荐。

采集了这些特征之后，我们就能够根据这些特征形成一个二维矩阵，一个维度是用户，另一个维度就是商品。矩阵里的值，可以根据不同的行为特征来定义。下面就是一个关于 3 位用户对 6 种商品喜好程度的例子。

```
> mx
      [,1] [,2] [,3]
[1,] 0.25 0.28 0.87
[2,] 0.11 0.62 0.96
[3,] 0.28 0.08 0.67
[4,] 0.31 0.75 0.95
[5,] 0.85 0.68 0.27
[6,] 0.22 0.78 0.53
```

通过这类矩阵的计算，就能实现诸如协同过滤这样的算法。不过，当用户行为数据规

<sup>①</sup> 这里只考虑网站内部能采集到的数据。如果和其他平台合作，可能会获得更为完整的用户行为数据。

模还很小的时候，会产生稀疏矩阵<sup>①</sup>，导致许多时候推荐的结果为零或很少。这时，常用的解决办法有如下几种：

- ❑ 将商品的维度换成分类维度。这是非常直观的方式，用户访问某个具体的商品概率可能非常低，但是访问该商品分类的可能性非常高。分类一般也是多个层级的，分类粒度越粗，所推荐的覆盖面就会越广，但是精确度也就会越差。
- ❑ 将商品维度换成商品聚类。根据商品的内容特征，将相似的商品聚为一组。可以认为是上面一种方法的扩展，不过其粒度通常小于分类，因此形成的推荐会更精准些。该方式也是内容特征和行为特征结合的一种。
- ❑ 将用户维度换成用户聚类。该方法类似于上一种方法，不过是根据用户的内容特征，将相似用户聚为一组。该方式也是内容特征和行为特征结合的一种。
- ❑ 拉长统计的时间窗口。例如，原本是观察最近一天的数据，改为观察最近一周的数据。这样某位用户访问某款商品的概率也会提升。

要注意这几种方式都是通过牺牲精确度来换取覆盖面的，和信息检索里的精度 / 召回率类似，需要取一个平衡点。

有了行为特征的矩阵后，我们来看看 Mahout 能做些什么。Mahout<sup>②</sup>实际上包含了很多推荐引擎模型的实现，大多都是源自基于用户、物品和 Slope-One 的技术。还有一些实验性的、初步的 SVD 矩阵的分解实现。可以通过如下几个 Java 语言的类来建立一个最简单的 Mahout 编程示例。

## 1. 数据模型 DataModel

对于用户偏好数据的压缩表示，具体又可分为<sup>③</sup>：

- ❑ FileDataModel：从文本加载用户 - 物品二维矩阵。
- ❑ JDBCDataModel：从关系型数据库加载用户 - 物品二维矩阵。
- ❑ GenericDataModel：从内容中加载用户 - 物品二维矩阵。
- ❑ RecommendedItem：推荐的物品，通常是以列表 (List) 的形式返回。

## 2. 基于用户的协同过滤

### (1) 推荐器 Recommender

基于用户推荐算法的核心实现部分，主要分为以下两种。

- ❑ GenericUserBasedRecommender：基于用户的推荐器，用户对物品的偏好用连续的值来表示。
- ❑ GenericBooleanPrefUserBasedRecommender：基于用户的无偏好值推荐器，用户对物品的偏好仅仅用 0 或 1 来表示。

① 如果在一个矩阵中，大多数的元素为 0，则称此矩阵为稀疏矩阵。

② 本文用的 Mahout 版本是 0.8。

③ 也有人实现了基于 HDFS 的 DataModel。



## (2) UserSimilarity

根据 DataModel 进行计算,用于衡量两个用户间的相似度。它是基于协同过滤的推荐引擎的核心部分,可以用来寻找某位用户的“近邻”。常见的计算指标在第 6 章中有介绍,具体又可分为如下的内容。

□ PearsonCorrelationSimilarity: 基于皮尔逊相关系数的相似度。

□ EuclideanDistanceSimilarity: 基于欧氏距离的相似度。

□ UncenteredCosineSimilarity: 基于夹角余弦的相似度。

□ LogLikelihoodSimilarity: 基于对数似然比的相似度。

□ SpearmanCorrelationSimilarity: 基于斯皮尔曼相关系数的相似度。

□ CityBlockSimilarity: 基于曼哈顿距离的相似度。

□ TanimotoCoefficientSimilarity: 基于谷本系数的相似度。

## (3) UserNeighborhood

对此, Mahout 提供了两种计算方式:

□ NearestNUserNeighborhood: 对每个用户取固定数量为 N 个的最近邻居。

□ ThresholdUserNeighborhood: 对每个用户基于一定的限制,取落在相似度阈值以内的所有用户为邻居。

## 3. 基于物品的协同过滤

### (1) 推荐器 Recommender

基于物品推荐算法的核心实现部分,主要分为以下 3 种:

□ GenericItemBasedRecommender: 基于物品的推荐器,用户对物品的偏好用连续的值来表示。

□ GenericBooleanPrefItemBasedRecommender: 基于物品的无偏好值推荐器,用户对物品的偏好仅仅用 0 或 1 来表示。

□ KnnItemBasedRecommender: 基于物品的 KNN 推荐算法。

### (2) ItemSimilarity

用于计算物品之间的相似度。具体计算指标可选类型参见上面的 UserSimilarity。

## 4. 其他推荐算法

□ SlopeOneRecommender: Slope 推荐算法。

□ SVDRecommender: SVD 推荐算法。

□ TreeClusteringRecommender: TreeCluster 推荐算法。

如果手头的数据中,用户的推荐结果有经过人为的打分,那么还能通过 Mahout 的评分器 RecommenderEvaluator 进行量化的评测。它有以下几种实现方式。

### (1) 测算分数型

计算推荐结果排序和人为打分的差距,具体分为如下两种形式。

□ AverageAbsoluteDifferenceRecommenderEvaluator：计算平均差值。

□ RMSRecommenderEvaluator：计算均方根差值。

## （2）测算准确性

可采用 RecommenderIRStatsEvaluator 衡量包括准确率、精度和召回率在内的指标，具体请参考第 7 章关于信息检索系统效果的评估。

为了支持海量数据，Mahout 提供了 RecommenderJob（类以 MapReduce 的方式）来实现协同过滤。该类将各种 Mapper 和 Reducer 组件连接起来，让 Hadoop 集群接手后面的一系列运算。

更多关于 Mahout 实现的推荐，以及在 Hadoop 上部署的细节，可以参考 Sean Owen 等人所著的《Mahout 实战》。

关于 Hadoop 部署的细节，可以参考陆嘉恒所著的《Hadoop 实战（第 2 版）》。

## 10.3.3 提供在线服务

R 和 Mahout 可以帮助我们实现多种模型，但是无法解决时效性问题。挖掘的算法都需要大量的计算，至少需要花上数分钟，甚至更久。前端用户访问的时候，绝对不可能等待这么长的时间。在这种情况下，可以预先进行离线挖掘，然后将批处理后的结果放入 Redis 或 Solr 集群，大幅提升访问的速度。

首先，看下如何将推荐结果存放到 Redis 中。Redis 的 Value 值支持多种数据类型的存放，考虑到推荐的内容大多数是一个列表，而且相似性 / 相关性（或者说是质量）由高到低有所不同，因此选择有序的列表（List）类型，以保证更高质量的推荐排在前面。图 10-5 展示了针对用户的推荐存储，Key 键是用户 ID，而 Value 值是给这个用户推荐的商品 ID 列表，适用于“个性化主页”或“用户中心”这样的栏位。图 10-6 展示了针对商品的推荐存储，Key 键是商品 ID，而 Value 值是和这款商品相似、相关的商品 ID 列表，使用了“看了此商品的用户还看了”、“看了此商品的用户最终买了”等栏位。

下面的 Redis<sup>①</sup>命令<sup>②</sup>展示了存放的示例，采用的是在列表右侧加入的方式。首先是对 UserA 和 UserB 进行存储：

```
127.0.0.1:6379> RPUSH UserA Product73 Product82
(integer) 2
127.0.0.1:6379> RPUSH UserB Product92 Product3 Product15
(integer) 3
```

然后是对 Product1 进行存储：

```
127.0.0.1:6379> RPUSH Product1 Product37 Product53 Product17 Product95
(integer) 4
```

① 本文的 Redis 使用的是 3.0.2 版本。

② 上面只是使用命令来进行示例，实际中可能需要编写代码或脚本来实现这种逻辑。

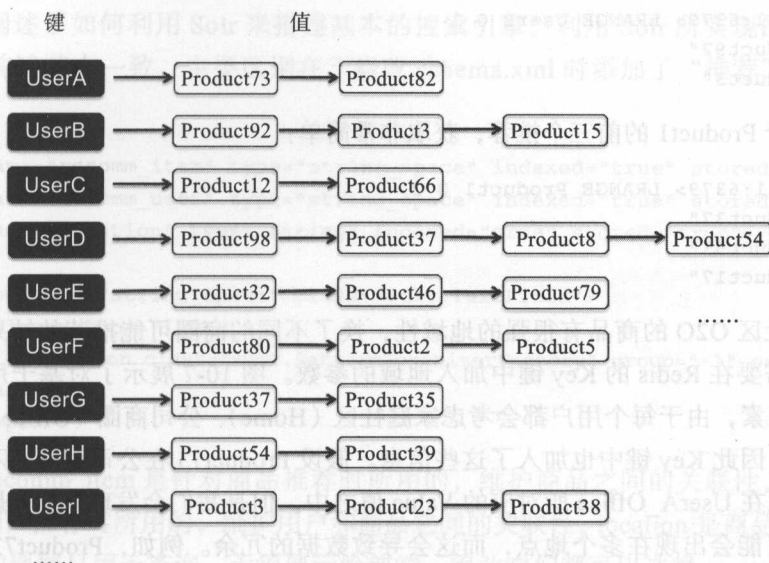


图 10-5 Redis 存放基于用户的推荐，键是用户 ID，值是推荐商品 ID 列表

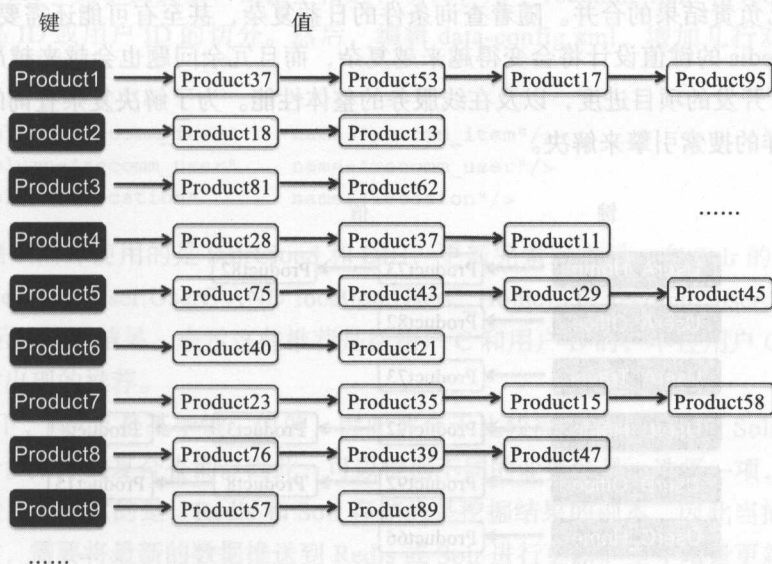


图 10-6 Redis 存放基于商品的推荐，键是商品 ID，值是推荐商品 ID 列表

下面分别获取对于 UserA 和 UserB 的前两个推荐：

```
127.0.0.1:6379> LRange UserA 0 1
```

```
1) "Product73"
```

```
2) "Product82"
```



```
127.0.0.1:6379> LRange UserB 0 1
1) "Product92"
2) "Product3"
```

以及对于 Product1 的前三个推荐，获取非常简单：

```
127.0.0.1:6379> LRange Product1 0 2
1) "Product37"
2) "Product53"
3) "Product17"
```

但是，社区 O2O 的商品有很强的地域性，换了不同的商圈可能推荐的结果就会完全不同，这时还需要在 Redis 的 Key 键中加入地域的参数。图 10-7 展示了对基于用户的推荐如何加入地域因素，由于每个用户都会考虑家庭社区（Home）、公司商圈（Office）和旅游地商圈（Travel），因此 Key 键中也加入了这些信息。假设 Product73 在公司商圈并不销售，那么它就不会出现在 UserA\_Office 所对应的 Value 值之中。但是我们会发现新的问题：有些快消或流行商品可能会出现在多个地点，而这会导致数据的冗余。例如，Product73 在 UserA 的家庭社区和旅游地商圈都有出现，Product92 和 Product8 在 UserB 的家庭社区和公司商圈也都有出现。而且，对于购物车里的栏位，可能推荐输入的不是单件商品，而是多件，那么开发者还要自己负责结果的合并。随着查询条件的日益复杂，甚至有可能还需要多条件复合，这样一来，Redis 的键值设计将会变得越来越复杂，而且冗余问题也会越来越严重，最终肯定会影响设计开发的项目进度，以及在线服务的整体性能。为了解决复杂查询的需求，可以引入 Solr 这样的搜索引擎来解决。

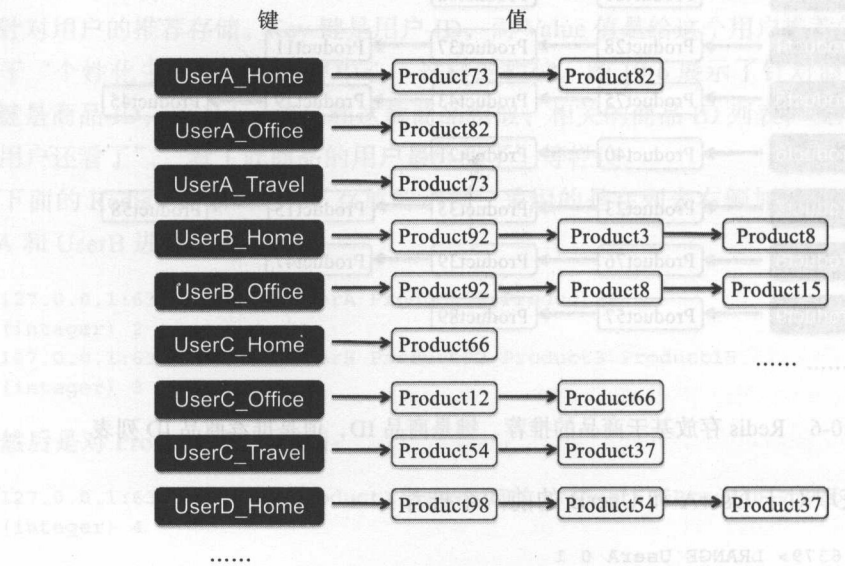


图 10-7 Redis 存放基于用户的推荐时，考虑到地域的因素

第 9 章阐述了如何利用 Solr 来搭建基本的搜索引擎。利用 Solr 所实现的推荐查询部分的流程和前述基本一致。主要区别在于修改 schema.xml 时添加了“推荐”字段, 代码如下:

```
<field name="recomm_item" type="string_space" indexed="true" stored="true" />
<field name="recomm_user" type="string_space" indexed="true" stored="true" />
<field name="location" type="string" indexed="true" stored="true" />

<fieldType name="string_space" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.PatternTokenizerFactory" group="-1" pattern=" " />
  </analyzer>
</fieldType>
```

其中, recomm\_item 是针对商品推荐时所用的, 维护商品之间的关联性, 而 recomm\_user 是针对用户推荐时所用的, 维护用户和商品之间的关联性, location 是商品销售的地域。因为关联性和地域只用于查询, 无须展示给前端, 因此它们都可以选择 stored=false, 用于节省存储空间。这里选择 stored=true, 是为了便于显示查询的结果。另外, 这里还自定义了“string\_space”的字段类型, 根据空格进行切词, 用于 recomm\_item 和 recomm\_user 字段中多个物品 ID 或用户 ID 的切分。然后, 编辑 data-config.xml, 增加几行对于 3 个新字段的映射:

```
<field column="recomm_item" name="recomm_item"/>
<field column="recomm_user" name="recomm_user"/>
<field column="location" name="location"/>
```

这里的测试同样使用的是 SolrCloud 和 DIH, 更新完索引之后, 在 Solr 的查询界面中, q 字段输入 (recomm\_user:UserC AND location:UserC\_Home) OR recomm\_user:UserB, 得到如图 10-8 所示的查询结果, 表示这些推荐是给用户 C 和用户 B 的, 不过用户 C 要求是在其家庭社区里才出现的推荐。

“总结一下, Redis 是基于键-值的, 因此适合于比较简单的查询, 而 Solr 是专业的搜索引擎, 适合多条件的复合查询。因此, 可以根据不同的业务场景, 选择一项, 或者两项同时配备。最后, 要注意的是, Redis 和 Solr 存储的是挖掘结果的副本, 因此当推荐的内容有所变化的时候, 需要将最新的数据推送到 Redis 或 Solr 进行更新。至于增量更新的机制, 今后有机会再做介绍。”

“好的, 已经很感谢了, 小明哥。今天收获颇丰, 框架基本都清晰了, 我们明天就开始动手实施自己的推荐引擎!”

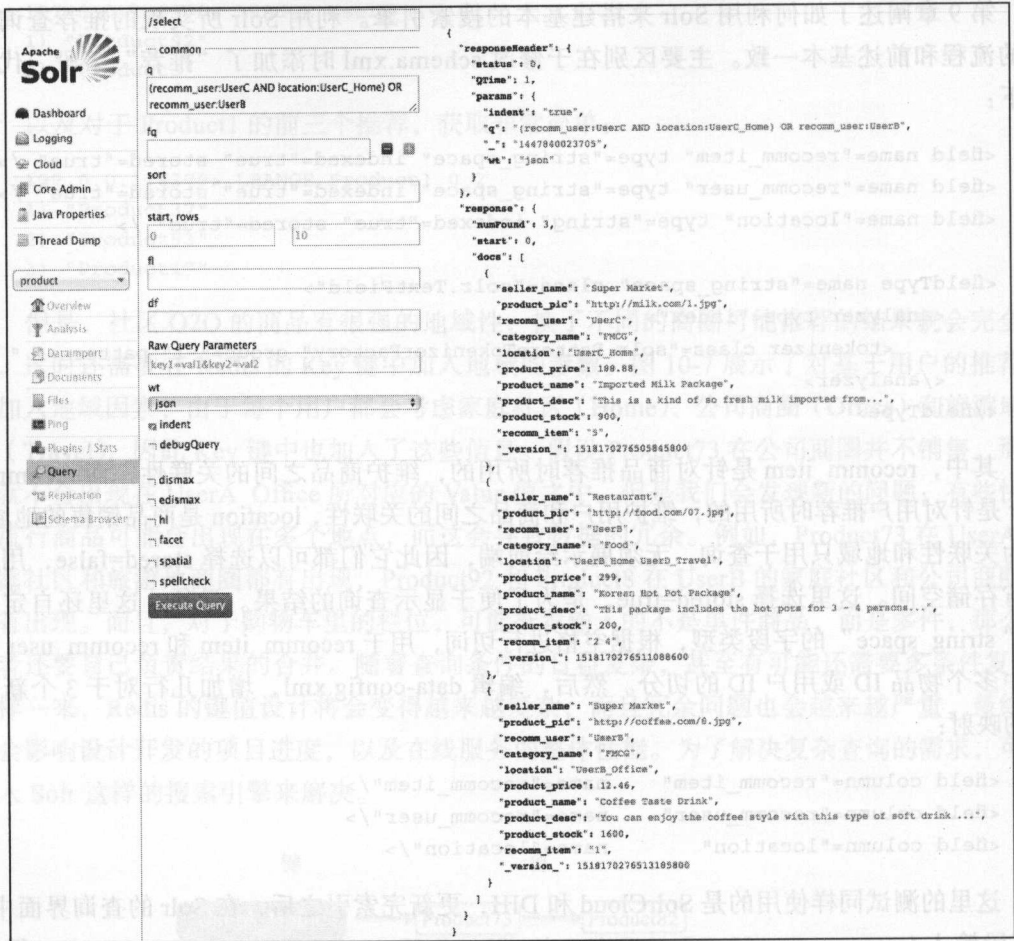
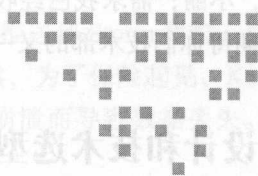


图 10-8 使用 Solr 支持推荐结果的查询，可以使用复杂的查询条件



图 10-7 Redis 存放基于用户的推荐时，考虑到地域的因素





## 第 11 章 Chapter 11

## 这样做效果如何

方案确定后，大宝团队的执行力是一流的，第一版的推荐系统很快就上线了，形成了分布在首页、用户中心、商品详情页、购物车结算等页面的多个栏位。由于连续攻克了搜索、推荐两大核心模块，管理层还特意为开发团队颁发了公司最高级别的荣誉：“总裁特别奖”。一时间，大宝的技术部门风光无限。这天，小丽找到了大宝，大宝很是得意，自以为小丽是来表示感谢的。没想到，小丽此行的真正目的不在于此。

## 11.1 业务需求

“大宝，感谢你帮助我们解决了用户提出的两个最大问题。”

“哪里，应该的，为公司、为顾客服务。”

“我今天来找你，其实是为了一件更重要的事情。”

“哦？请讲。”大宝心里不禁犯嘀咕，最大的两个痛点都解决了，还能有什么呢？

“随着系统和业务的日趋完善，我们站点的访问量越来越大。公司的高层和业务都很关心的是，这些用户来到我们的站点之后，是如何表现的？又有怎样的消费习惯？具体一些就是，有多少人看过我们的促销页？多少人使用了搜索或推荐的功能？搜索和推荐的转化率又是如何的呢？目前我们采用的都是第三方统计软件，但是粒度太粗了，没法跟踪到一些细节。举个例子，我们很想知道从搜索结果页面点击进入商品详情页面的用户占比有多少？在搜索结果页直接添加购物车的用户占比又有多少？第三方的结果都没有办法支持。”

“让我想想……”大宝的脑子马上开始飞速地运转。确实，目前用户在网站上的行为都没有被很好地加以利用。如果能做出一套强有力的行为日志收集和分析系统，那么既能评估现有的功能模块，证明我们辛勤工作的价值，还能为将来的数据挖掘准备更多的素材，何乐

而不为？“好的，小丽，需求我已经收到，我们会尽快设计方案。”

“太棒了，期待你们技术部的又一次大作！”

## 11.2 产品设计和选型

大宝和团队成员沟通了业务的需求，有人提出“是不是可以先人工评估下各个系统模块的表现？”大宝回想起之前小明确实有提到过，评估系统的效果有很多离线的方式。例如根据人工标注好的数据集，对信息检索系统的精度和召回率，以及评测分类系统的准确率等进行评测。还有通过用户调研的形式，邀请用户或专家来判定系统是否有效。不过，这些都需要太多的人工干预，肯定需要不少的运营和财务成本，而且时间上也会拖延得比较久。因此，大宝最终还是决定设计一个自动化的数据采集、存储和分析系统，这样既可以快速地收集用户的反馈，而且又可以为线上进行 AB 测试提供可能。主要模块设计如下。

□ 采集：一些经典的 Web 服务器（例如 Apache Tomcat）已经能够收集很多对于页面的访问日志了。不过，由于需要考虑更细的粒度，这里应该设计一套前端采集的代码，用于收集对于某个栏位、坑位，甚至是按钮的点击等。但现在前端的服务器是多个分散的集群，如何将这些数据整合起来会是个大问题。不过，或许可以使用 Flume，它支持在系统中定制各类数据发送方，同时还支持对数据进行简单的处理，然后写到各种可定制的数据接受方。此外，Flume 还有一大优势，即它是集群化管理的，便于水平扩展，即使需要采集的应用很多，我们也不用担心收集系统无法承受。

□ 存储：从精确性角度而言，行为数据不像银行交易系统要求得那么严格。从内容的规范性来看，行为数据的格式可能会随着业务需求的变化而不断变化，且包含着较高的不确定性。另外，用户流量将来还会不断地增长，对于数据规模的要求反而会更高。综合这三点，扩展性良好的 Hadoop HDFS 是不错的选择，它可以非常方便地存储海量的非结构化数据。

□ 分析：数据放入 HDFS 后，自然会想到利用 MapReduce 的框架来分析。不过，为了一般的统计需求去写大量的 MapReduce 代码实在是得不偿失。对此，Hive 能够帮助节省不少的时间和精力，大幅提升整体效率。Hive 是建立在 Hadoop 基础之上的数据仓库工具，可以存储、查询和分析存储在 HDFS 中的大规模数据。它提供的类 SQL 查询语言 HiveQL，对于熟悉 SQL 的用户而言，入门和使用也非常方便。

此时有人提出，虽然有了基本框架，不过 Flume、HDFS 和 Hive 擅长的都是对实时性要求不高的批处理，万一有些数据报表需要非常及时地生成，又该如何处理呢？大宝真心觉得这是个绝好的问题，按照小明之前的指导，他觉得可以考虑选择类似 Kafka 的消息机制，它提供了比 Flume 更敏捷的采集速度，避免过度频繁的批处理操作。如果还要进一步提升数据生产的速度，那么 Storm 应该也是一个不错的选择，这样的流式计算可以保证在第一时间内获得统计的结果。经过这全盘的思考，最终大宝初步敲定了系统的架构，如图 11-1 所

示。在这个架构中,Flume 和 Kafka 可以同时获取数据,不过 Flume 侧重于非及时性的数据,而 Kafka 侧重于及时性数据。Kafka 如果将数据进一步推送到 Storm 集群,就可以满足更高的实时性要求,例如实时数据仪表盘这样的应用。当然,为了保险起见,Kafka 也会将实时消息推送到 HDFS 进行持久化存储,以免 Storm 集群崩溃而导致数据丢失。进入 HDFS 的数据就交由 Hive 负责产生批量的报表。

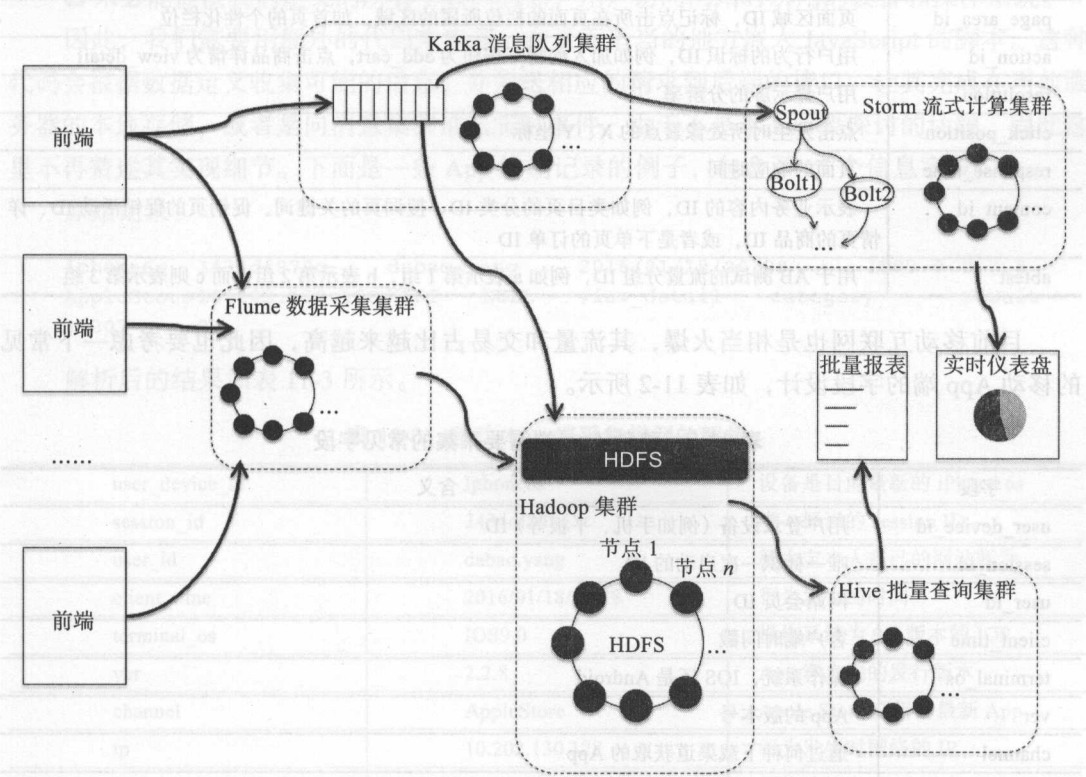


图 11-1 收集、存储和分析用户行为数据的框架

### 11.3 实现方案

#### 11.3.1 行为数据的定义和记录

在收集数据之前,必不可少的步骤仍然是定义我们需要收集什么。考虑到 O2O 电商的业务需求,常见的 PC 端字段设计如表 11-1 所示。



表 11-1 PC 端需要采集的常见字段

字段	含义
referrer_url	当前页的来源页（上一页）的 URL
request_url	当前界面的 URL
forward_url	即将跳转页面的 URL
user_id	网站会员 ID，可以通过用户登录时植入 Cookie 实现
page_area_id	页面区域 ID，标记点击所在页面的栏位所属的区域，如首页的个性化栏位
action_id	用户行为的标识 ID，例如加入购物车按钮为 add_cart，点击商品详情为 view_detail
resolution	用户显示屏的分辨率
click_position	点击发生时所处像素点的 X、Y 坐标
response_time	页面的响应时间
content_id	表示业务内容的 ID，例如类目的分类 ID、搜词页的关键词、促销页的促销活动 ID、详情页的商品 ID，或者是下单页的订单 ID
abtest	用于 AB 测试的流量分组 ID，例如 a 表示第 1 组，b 表示第 2 组，而 c 则表示第 3 组

目前移动互联网也是相当火爆，其流量和交易占比越来越高，因此也要考虑一下常见的移动 App 端的字段设计，如表 11-2 所示。

表 11-2 移动 App 端需要采集的常见字段

字段	含义
user_device_id	用户登录设备（例如手机、平板等）ID
session_id	唯一标识一次启动的 ID
user_id	网站会员 ID
client_time	客户端时间戳
terminal_os	操作系统，IOS 还是 Android
ver	App 的版本号
channel	通过何种下载渠道获取的 App
ip	访问时的 IP 地址
network	联网方式，2G、3G、4G 还是 WiFi
gps	用户允许的情况下，所记录的地理经纬度
action_id	用户行为的标识 ID，例如加入购物车按钮为 add_cart，点击商品详情为 view_detail
page_id	当前页面类型的 ID，例如搜词页为 search，类目页为 category，促销页为 promotion，详情页为 detail
page_area_id	页面栏位的 ID，例如搜索结果 result，推荐栏位为 recommend
content_id	表示业务内容的 ID，例如类目的分类 ID、搜词页的关键词、促销页的促销活动 ID、详情页的商品 ID，或者是下单页的订单 ID
abtest	用于 AB 测试的流量分组 ID，例如 a 表示第 1 组，b 表示第 2 组，而 c 则表示第 3 组

虽然有些相类似的字段，但是 App 还有些特殊的收集需求，例如移动设备的情况、访问的网络环境、用户的地理位置等，这些信息对于精准化的分析同样重要。当然，上述只是

最基础的设计，可能还需要根据实际需求制定更为详尽的字段列表。有了数据的定义，就可以着手进行记录<sup>①</sup>了。页面的浏览通常都会记录在 Web 服务器的日志中，例如 Apache 的 Tomcat。但是，这种记录至少有两点不足：

❑ 只能记录页面级别的，无法记录某个页面上更为细节的操作，比如是否点击了“加入购物车”的按钮。

❑ 未必能提供我们定义的所有字段。例如，用户访问网站时所用的设备和操作系统。

因此，我们需要用额外的代码来实现，在页面适当的地方嵌入 JavaScript 的脚本。这种代码会根据数据定义收集可能的信息，并发送相应的请求到后端的接口，让其完成在当前服务器的本地存储，或者是向消息集群推送实时事件。由于不是本文所要探讨的话题，因此这里不再赘述其实现细节。下面是一条 App 终端记录的例子，包含了 15 个信息字段。

原始记录：

```
iphone6s 3434df878g dabao.yang 2016/01/18/20/28 IOS9.0 2.2.8
AppleStore 10.202.130.128 WIFI NULL view_detail category result
32207 2
```

解析后的结果如表 11-3 所示。

表 11-3 移动 App 端采集样例的解析

user_device_id	Iphone6s	设备是目前最新的 iPhone 6s
session_id	3434df878g	唯一标识的 session ID
user_id	dabao.yang	杨大宝本人自己的网站账号
client_time	2016/01/18/20/28	行为发生的时间
terminal_os	IOS9.0	操作系统为 9.0 版本的 IOS
ver	2.2.8	App 客户端的发行版本
channel	AppleStore	Apple Store 获得的最新 App
ip	10.202.130.128	行为发生时网络的 IP
network	WIFI	网络连接方式
gps	NULL	大宝没有打开 GPS 定位
action_id	view_detail	查看详情的行为
page_id	category	发生在类目页
page_area_id	result	类目页中的搜索结果
content_id	32207	类目 ID
abtest	2	进行 AB 测试，属于第 2 组流量

通过 JavaScript 完成数据记录后，开发团队面临的问题是：我们所要的数据分散在多个服务器上。形成这个困境的主要原因是：出于负载均衡的需要，通常会有很多服务器负责处

① 这里的“记录”是指将用户行为按照数据定义生成并存储在本机上，而 Flume 的收集是指将多台机器上的日志传输到统一的存储系统，两者有所不同。

理前端用户的访问，每台机器只要处理了用户的请求，就会产生对应的行为数据。不过，可以利用 Flume 和 Hadoop 来解决该问题。

### 11.3.2 Flume 和 HDFS 的集成

Flume 可以向多个持久化存储中写入数据，这里选择 Hadoop 的 HDFS，因此首先要建立 Hadoop<sup>①</sup> 集群。在各个预备建立 Hadoop 的机器节点之间配置 SSH 的相互信任，然后修改 conf 目录中的主要配置文件 core-site.xml，代码如下：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://10.200.45.2:9000</value>
    <description>Name Node</description>
  </property>

  <property>
    <name>hadoop.tmp.dir</name>
    <value>/Users/dabao.yang/logs</value>
    <description>A base for log files.</description>
  </property>
</configuration>
```

其中，fs.default.name 指定了命名节点（Name Node）的 IP 和端口，而 hadoop.tmp.dir 指定了 HDFS 数据存放的目录。

修改配置文件 mapred-site.xml，代码如下：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>10.200.45.2:9001</value>
    <description>The host and port that the MapReduce job tracker runs at.
  </description>
  </property>
```

① 本书使用的 Hadoop 版本是 1.2.1，集群采用 Master Slave 机制。



```
</configuration>
```

其中, `mapred.job.tracker` 设置了工作跟踪节点 (Job Tracker), 主要用于在 MapReduce 计算时分配任务。

然后修改 `masters` 文件的内容如下, 将该节点作为主节点, 也就是命名节点和工作跟踪节点:

```
10.200.45.2
```

再修改 `slaves` 文件的内容如下, 除了主节点, 还包括一个从节点:

```
10.200.45.2
```

```
10.200.60.103
```

配置完毕后, 在主节点上通过如下命令格式化 HDFS:

```
hadoop namenode -format
```

然后在主节点上通过 Hadoop bin 目录中的如下命令启动 Hadoop 集群:

```
start-all.sh
```

如果要关闭集群, 则用 Hadoop bin 目录中的如下命令:

```
stop-all.sh
```

启动集群后, 可通过 `http://10.200.45.2:50070/dfshealth.jsp` 来查看集群的整体状况, 如图 11-2 所示, 其中包括两个节点、存储路径, 以及磁盘空间的总使用率等信息。

Browse the filesystem		
Namenode Logs		
Cluster Summary		
7 files and directories, 1 blocks = 8 total. Heap Size is 245.5 MB / 889 MB (27%)		
Configured Capacity	:	464.73 GB
DFS Used	:	16.03 KB
Non DFS Used	:	391.36 GB
DFS Remaining	:	73.37 GB
DFS Used%	:	0 %
DFS Remaining%	:	15.79 %
Live Nodes	:	2
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0
NameNode Storage:		
Storage Directory	Type	State
/Users/dabao.yang/logs/dfs/name	IMAGE_AND_EDITS	Active
显示版本: Apache Hadoop release 1.2.1		

图 11-2 集群整体状况

点击图 11-2 中的“Live Nodes”，可以看到两个节点的进一步信息，如图 11-3 所示。

[Browse the filesystem](#)  
[Namenode Logs](#)  
[Go back to DFS home](#)

Live Datanodes : 2

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
10.200.45.2	2	In Service	231.77	0	183.6	48.16	0		20.78	1
10.200.60.103	0	In Service	232.96	0	207.78	25.18	0		10.81	1

This is Apache Hadoop release 1.2.1

图 11-3 两个节点各自磁盘空间的使用率信息

点击图 11-3 中的“Browse the filesystem”，可以看到 HDFS 中文件的情况，如图 11-4 所示。由于集群刚刚启动，目前没有分布式文件，内容都是空的。唯一的 mapred 是系统生成的，用于将来的 MapReduce 计算。

Contents of directory /Users/dabao.ya

Goto : /Users/dabao.yang/logs go

Go to parent directory

Name	Type	Size	Replication	Block Size
mapred	dir			

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.2.1

图 11-4 浏览 /Users/dabao.yang/logs，没有看到任何文件

目前看来 Hadoop 集群的运行是正常的，接下来，让 Flume<sup>Ⓐ</sup>收集不同机器上的日志数据，并存储到 HDFS 中。修改 Flume conf 目录中的 flume-conf.properties，代码如下：

```
#agent 的名称为 o2oagent
#o2oagent 的 source、channel 和 sink 分别是 s1、c1 和 k1，具体定义见后面
o2oagent.sources = s1
o2oagent.channels = c1
o2oagent.sinks = k1
```

Ⓐ 这里使用 Flume 的版本是 1.5.2。

```

#s1 的定义
# 类型是从文件目录中读取
o2oagent.sources.s1.type = spooldir
#Web 服务器存放日志的目录, 从这里收集
o2oagent.sources.s1.spoolDir = /webserver/accesslog
# 源头 s1 连接的通道是 c1
o2oagent.sources.s1.channels = c1
# 每条日志的最大字符长度
o2oagent.sources.s1.deserializer.maxLineLength = 4096

# 通道 c1 的定义
# 类型是内存
o2oagent.channels.c1.type = memory
# 最大容量
o2oagent.channels.c1.capacity = 20000
# 事务容量
o2oagent.channels.c1.transactionCapacity = file

# 沉淀器 k1 的定义
# 类型是 HDFS, 数据将放入其中
o2oagent.sinks.k1.type = hdfs
#HDFS 的存储路径
o2oagent.sinks.k1.hdfs.path = hdfs://10.200.45.2:9000/Users/dabao.yang/logs
# 日志记录在收集过程中, 加入必要的前缀、后缀
o2oagent.sinks.k1.hdfs.filePrefix = o2o-event
o2oagent.sinks.k1.hdfs.fileSuffix = .log
# 文件类型和写入类型
o2oagent.sinks.k1.hdfs.fileType = DataStream
o2oagent.sinks.k1.hdfs.writeType = text
# 超时设置
o2oagent.sinks.k1.hdfs.request-timeout = 30000
# 定期批量保存的参数
o2oagent.sinks.k1.hdfs.rollSize = 61440000
o2oagent.sinks.k1.hdfs.rollCount = 500000
o2oagent.sinks.k1.hdfs.rollInterval = 300
# 沉淀器 k1 连接的通道是 c1, c1 将 s1 和 k1 连接起来
o2oagent.sinks.k1.channel = c1

```

在每台需要收集行为日志的机器上部署 Flume 并配置相应的 flume-conf.properties, 需要注意的是, 每台机器上要收集的日志目录可能会有所不同。最后, 通过 Flume bin 目录的 flume-ng 命令来启动每台机器上的 Flume 代理。

```
flume-ng agent -n o2oagent -c conf -f conf/flume-conf.properties -Dflume.root.logger=INFO,console
```

其中, -n o2oagent 指定了代理的名称, -c conf -f conf/flume-conf.properties 指定了配置文件, 最后的 -Dflume.root.logger=INFO,console 指定了在 Console 命令行进行信息输出的级别。一旦 Flume 的代理 o2oagent 启动成功, 当我们再次刷新 HDFS 的 /Users/dabao.yang/

logs 目录时，就会发现有不少 log 日志正在写入，如图 11-5 所示。这里 o2o-event 是之前定义的前缀，.log 是定义的后缀，.log.tmp 表示正在写入的最新数据文件。过几分钟，再次刷新同一目录，发现又新增了更多的文件，如图 11-6 所示。

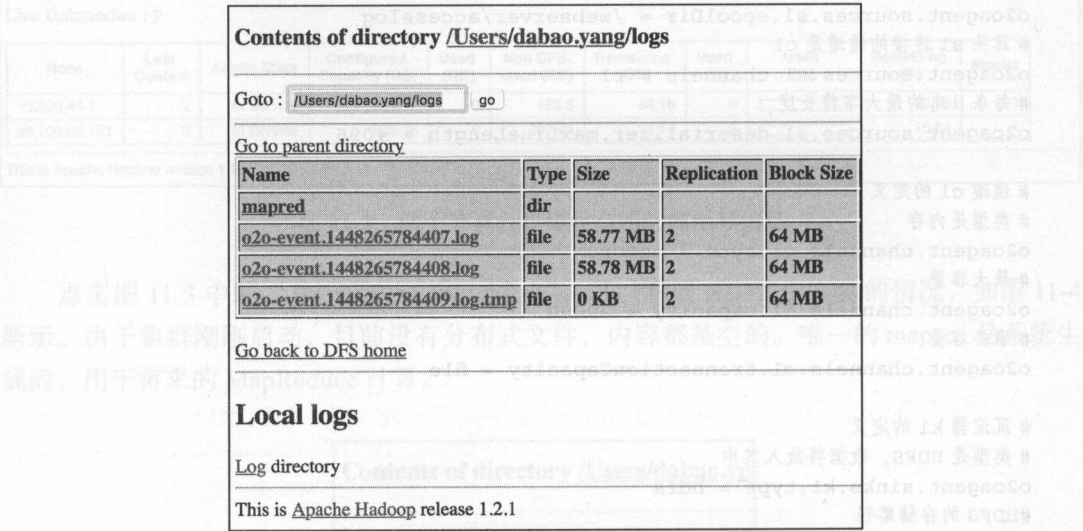


图 11-5 浏览 /Users/dabao.yang/logs，看到有新的文件正在写入

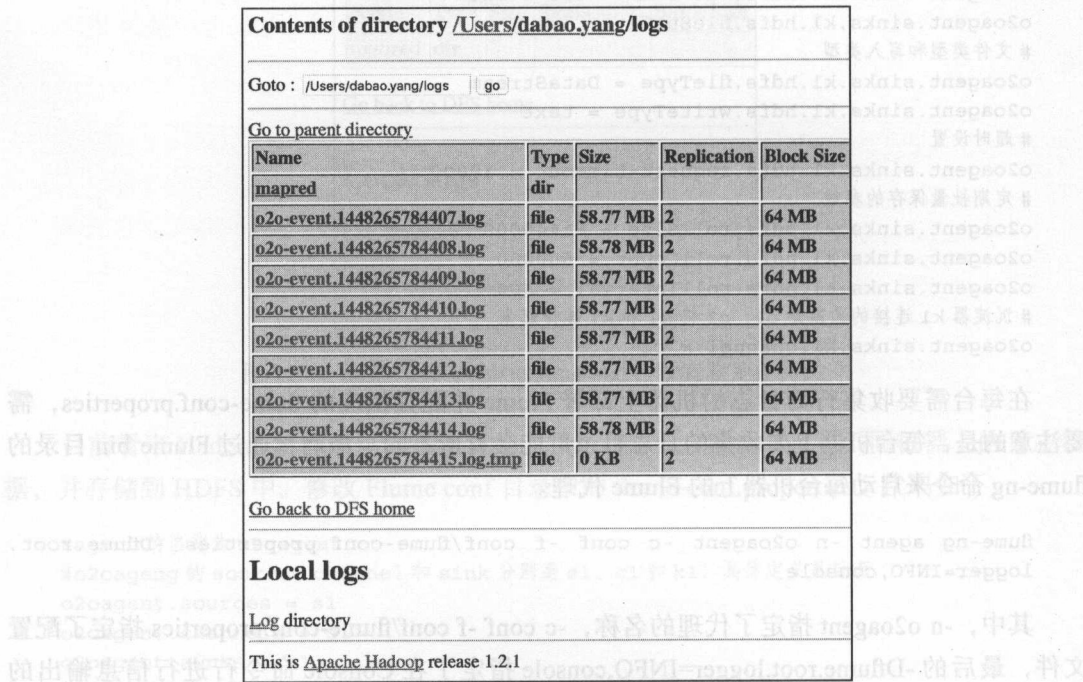


图 11-6 刷新 /Users/dabao.yang/logs，看到不断有新的文件在写入



随机挑选一个日志文件，点击打开，可以看到如图 11-7 所示的内容，每一行都是一条访问的日志记录，表明数据已经采集成功。从图 11-8 中可以看到，两个节点在截屏时都已经使用了 0.94GB 的磁盘空间，用于存储这些访问日志。

**File: /Users/dabao.yang/logs/o2o-event.1448265784408.log**

Goto:

[Go back to dir listing](#)  
[Advanced view/download options](#)

[View Next chunk](#)

```

10.201.128.253|2014-06-18T21:57:08+08:00|1403099828.722|search02.idc1.fn|4973168|GET / HTTP/1.1|200|156387|-|Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win84; x64; Trident/5.0)|110.75.105.251
10.201.128.253|2014-06-18T21:57:11+08:00|1403099831.573|search02.idc1.fn|4973328|GET /search/autocomplete?term=%E6%B9%BF%E5%B7%BE
HTTP/1.1|200|427|http://search.feiniu.com/search/?q=%E6%B4%B1%E4%BA%91%E6%B9%BF%E5%B7%BE|Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML,
like Gecko) Chrome/21.0.1180.89 Safari/537.1|124.77.93.62
10.201.128.253|2014-06-18T21:57:11+08:00|1403099831.884|search02.idc1.fn|4973285|GET /search/?q=%E8%84%89%E5%8A%A8
HTTP/1.1|200|166353|http://www.mizone.cc/ime/?mod=gift|Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_1 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko) Mobile/11D201
MicroMessenger/5.3.1|15.63.42.243
10.201.128.253|2014-06-18T21:57:15+08:00|1403099835.357|search02.idc1.fn|4973328|GET /search/autocomplete?term=fen
HTTP/1.1|200|14|http://search.feiniu.com/search/?q=%E6%B4%B1%E4%BA%91%E6%B9%BF%E5%B7%BE|Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like
Gecko) Chrome/21.0.1180.89 Safari/537.1|124.77.93.62
10.201.128.253|2014-06-18T21:57:15+08:00|1403099835.592|search02.idc1.fn|4973294|GET /search/?q=%E6%B4%97%E6%B4%B1%E7%B2%BE
HTTP/1.1|200|199165|http://www.feiniu.com/?utm_source=Baidu&utm_medium=Brand&utm_content=D11&utm_campaign=1414w1_index&ref=mk_Baidu_Brand|Mozilla/5.0 (iPad;
CPU OS 7_1_1 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko) Version/7.0 Mobile/11D201 Safari/9537.53|14.155.253.23
10.201.128.253|2014-06-18T21:57:16+08:00|1403099838.283|search02.idc1.fn|4973285|GET /search/autocomplete?term=feng
HTTP/1.1|200|14|http://search.feiniu.com/search/?q=%E6%B4%B1%E4%BA%91%E6%B9%BF%E5%B7%BE|Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like
Gecko) Chrome/21.0.1180.89 Safari/537.1|124.77.93.62
10.201.128.253|2014-06-18T21:57:18+08:00|1403099838.059|search02.idc1.fn|4973285|GET /search/autocomplete?term=fengli
HTTP/1.1|200|14|http://search.feiniu.com/search/?q=%E6%B4%B1%E4%BA%91%E6%B9%BF%E5%B7%BE|Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like
Gecko) Chrome/21.0.1180.89 Safari/537.1|124.77.93.62
10.201.128.253|2014-06-18T21:57:18+08:00|1403099838.459|search02.idc1.fn|4973285|GET /search/autocomplete?term=fenglis
HTTP/1.1|200|14|http://search.feiniu.com/search/?q=%E6%B4%B1%E4%BA%91%E6%B9%BF%E5%B7%BE|Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like
Gecko) Chrome/21.0.1180.89 Safari/537.1|124.77.93.62
10.201.128.253|2014-06-18T21:57:18+08:00|1403099838.963|search02.idc1.fn|4973285|GET /search/autocomplete?term=%E5%87%A4%E6%A2%A8%E9%85%A5
HTTP/1.1|200|88|http://search.feiniu.com/search/?q=%E6%B4%B1%E4%BA%91%E6%B9%BF%E5%B7%BE|Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like

```

[Download this file](#)  
[Tail this file](#)

Chunk size to view (in bytes, up to file's DFS block size):

**Total number of blocks: 1**  
-514675165939341288: 10.200.56.187:50010 10.200.56.217:50010

[Go back to DFS home](#)

**Local logs**

[Log directory](#)

This is Apache Hadoop release 1.2.1

图 11-7 打开其中一个 log 文件，看到访问日志已经获取成功

[Browse the filesystem](#)  
[NameNode Logs](#)  
[Go back to DFS home](#)

**Live Datanodes : 2**

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
10.200.45.2	2	In Service	231.77	0.94	183.82	47	0.41	<div></div>	20.28	18
10.200.60.103	1	In Service	232.96	0.94	211.07	20.95	0.4	<div></div>	8.99	18

This is Apache Hadoop release 1.2.1

图 11-8 再次查询两个节点的磁盘使用情况

关于 Hadoop 部署的细节，可以参考陆嘉恒所著的《Hadoop 实战（第2版）》。

关于 Flume 部署的细节，可以参考 Hari Shreedharan 所著的《Flume：构建高可用、可扩展的海量日志采集系统》。

### 11.3.3 通过 Hive 进行分析

在 Flume 顺利地将数据传送到 Hadoop 的 HDFS 中之后，就可以利用 Hive<sup>①</sup> 进行快捷地统计分析了。在前面 Hadoop 集群启动完毕后，在主节点上设置正确的环境变量 HADOOP\_HOME，最后执行 Hive bin 目录里的 hive 就行了。

然后利用如下命令来建立外部表：

```
CREATE EXTERNAL TABLE accesslog(frontendip STRING, time STRING, timestamp
DOUBLE, server STRING, sessionid STRING, url STRING, status INT, guid STRING,
reserve STRING, browser STRING, ip STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|';
```

执行成功后显示如下耗时信息：

```
OK
Time taken: 0.203 seconds
```

再将 Flume 收集到 HDFS 的数据，加载到创建的 Hive 表中，选中 o2o-event.1448-265784408.log 这个日志文件：

```
load data inpath '/Users/daobao.yang/logs/o2o-event.1448265784408.log' into
table accesslog;
```

执行成功后显示耗时情况、处理文件的大小等信息，结果如下：

```
Loading data to table default.accesslog
Table default.accesslog stats: [numFiles=1, numRows=0, totalSize=61630430,
rawDataSize=0]
OK
Time taken: 1.161 seconds
```

之后就可以使用类 SQL 语言在 Hive 上进行查询了，可以看到目前加载了多少条事件记录：

```
SELECT COUNT(*) FROM accesslog;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
```

① 本书使用的 Hive 的版本是 0.13.1。

```

set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapred.reduce.tasks=<number>
Starting Job = job_201601070402_0002, Tracking URL = http://101.58.220.128:50030
/jobdetails.jsp?jobid=job_201601070402_0002
Kill Command = /Users/daobao.yang/bigdata/hadoop-1.2.1/libexec/./bin/hadoop
job -kill job_201601070402_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-01-07 04:44:21,518 Stage-1 map = 0%, reduce = 0%
2016-01-07 04:44:25,542 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.56 sec
2016-01-07 04:44:34,599 Stage-1 map = 100%, reduce = 33%, Cumulative CPU 1.56 sec
2016-01-07 04:44:35,611 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.68 sec
MapReduce Total cumulative CPU time: 2 seconds 680 msec
Ended Job = job_201601070402_0002
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 2.68 sec HDFS Read: 50815754 HDFS
Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 680 msec
OK
1903142
Time taken: 17.501 seconds, Fetched: 1 row(s)

```

从中可以看出，Hive 启动了 Hadoop 的 MapReduce 任务来执行查询 “SELECT COUNT(\*) FROM accesslog”，最后执行得到的结果是共有 1,903,142 条记录。

如果想得知 190 万次页面访问中有多少次是搜索的访问，那么可根据 URL 是否带有 search 字样来判断，执行查询 “SELECT COUNT(\*) FROM accesslog WHERE url LIKE '%search%’”，代码如下：

```

SELECT COUNT(*) FROM accesslog WHERE url LIKE '%search%';
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapred.reduce.tasks=<number>
Starting Job = job_201601070402_0003, Tracking URL = http://101.58.220.128:
50030/jobdetails.jsp?jobid=job_201601070402_0003
Kill Command = /Users/daobao.yang/bigdata/hadoop-1.2.1/libexec/./bin/hadoop
job -kill job_201601070402_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-01-07 04:50:01,738 Stage-1 map = 0%, reduce = 0%
2016-01-07 04:50:06,776 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.16 sec
2016-01-07 04:50:14,813 Stage-1 map = 100%, reduce = 33%, Cumulative CPU 2.16 sec
2016-01-07 04:50:15,816 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 3.27 sec
MapReduce Total cumulative CPU time: 3 seconds 270 msec
Ended Job = job_201601070402_0003

```

```

MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 3.27 sec HDFS Read: 50815754 HDFS
Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 270 msec
OK
748726
Time taken: 18.732 seconds, Fetched: 1 row(s)

```

最后执行的结果是 748,726 条记录，因此搜索占比整体的访问流量是  $748,726 / 1,903,142 = 39.34\%$ 。以此类推，还可以完成很多 KPI 指标的计算，例如各渠道流量绝对值、流量占比、到详情页或添加购物车的转化率等。由于 Hive 支持很多类 SQL 的操作，因此入门简单，功能强大，对于批量处理的报表而言这绝对是一大利器。Hive 的更多使用方法可以参考 Edward Capriolo 等人所著的《Hive 编程指南》。

如果将流量进行分组，并在行为日志中记录分组 ID，那么还可以进行在线的 AB 测试。对于每组流量，我们可以设置不同的算法、不同的流程或不同的交互，然后根据每组收集到的数据，采用 Hive 来分析得出 KPI 报表，用于对比不同方法所带来的效果。最后，有两个常见的误区需要注意。

❑ 流量分组策略：最简单的策略是进行随机分配。不过，有些应用对于同一个用户而言，需要保持前后一致。例如，我们不能让用户一会看到新版的页面，一会又看到旧版的页面，这样他们就会感到非常迷茫，不知道发生了什么事情，严重的甚至会让无法将购物流程继续下去。更好的做法是根据用户账户或 Session ID 等唯一标识进行随机分配，并且保持 Session Sticky，保证至少在每次访问会话期间访问的内容是一致的。

❑ 统计意义：在进行不同流量分组的对比试验时，还需要注意第 7 章所介绍的统计意义，避免偶然性导致的不确定性结论。

### 11.3.4 Kafka 和 Storm 的集成

Flume、HDFS 和 Hive 的搭配对于批量处理来说是比较合适的。可是，对于实时性更强的需求，例如实时仪表盘这样的应用，必须要加速整体的数据处理速度。这里采用的是 Kafka 消息机制和 Storm 流式处理的集成方式。

首先，建立 ZooKeeper<sup>①</sup> 集群，用于 Kafka 等其他集群的分布式管理。假设当前一共有三台机器，IP 分别是 10.200.5.88、10.200.5.66 和 10.200.5.32。进入 ZooKeeper 的 conf 目录，修改 zoo.cfg 文件，加入服务器节点的 IP 和端口号：

```

server.1 = 10.200.5.88:2888:3888
server.2 = 10.200.5.66:2888:3888
server.3 = 10.200.5.32:2888:3888

```

① 本书使用的 ZooKeeper 版本是 3.4.6。



这些配置内容要在所有服务器节点上生效。在 zoo.cfg 中有一项配置是 dataDir，可在其所指定的数据目录下，创建文件 myid，文件内容为一个正整数，用来唯一标识当前的服务器节点，因此不同机器的数值不能相同。约定俗成是从 1 开始递增标识，以方便记忆和管理，例如：

```
10.200.5.88      1
10.200.5.66      2
10.200.5.32      3
```

最后，在每台服务器上，进入 ZooKeeper 的 bin 目录，通过下面的命令逐台启动 ZooKeeper：

```
zkServer.sh start
```

返回的结果显示启动成功：

```
JMX enabled by default
Using config: /Users/daobao.yang/bigdata/Zookeeper-3.4.6/bin/./conf/zoo.cfg
Starting Zookeeper ... STARTED
```

然后依次查看 3 台机器的状态：

```
zkServer.sh status
```

其中两台返回的结果表明它们是跟随者 (Follower)，如下：

```
JMX enabled by default
Using config: /Users/daobao.yang/bigdata/Zookeeper-3.4.6/bin/./conf/zoo.cfg
Mode: follower
```

另 1 台的返回结果表明它是领袖 (Leader)，如下：

```
JMX enabled by default
Using config: /Users/daobao.yang/bigdata/Zookeeper-3.4.6/bin/./conf/zoo.cfg
Mode: leader
```

这样 3 台机器组成的 Zookeeper 集群搭建完成，更多关于 ZooKeeper 部署的细节可以参考陆嘉恒所著的《Hadoop 实战 (第 2 版)》。

接下来可以在其基础上创建 Kafka 集群，用于发送实时消息。首先在 ZooKeeper 的 bin 目录里执行 zkCli.sh，创建 Kafka 的路径：

```
zkCli.sh -server 10.200.5.88:2181
create /kafka ''
```

然后依次进入每个 Kafka 机器节点上 Kafka 的 config 目录，修改 server.properties 相应的条目，代码如下：

```
broker.id=0
```

```
Zookeeper.connect=10.200.5.88:2181,10.200.5.66:2181,10.200.5.32:2181/kafka
```

需要注意的是，broker.id 和 ZooKeeper 的 myid 类似，它在每台机器上都保持不一样，通常是从 0 开始逐个递增的非负整数。修改配置完毕，就可以按照下面的命令依次启动每台服务器上的 Kafka 服务了：

```
nohup kafka-server-start.sh /Users/daobao.yang/bigdata/ kafka_2.11-0.8.2.0/
config/server.properties &
```

然后创建消息的 topic 名，即 accesslog-topic，这里使用了两个副本和 6 个分区。需要注意的是，副本参数的数值不能大于 Kafka 服务的数量，否则启动可能会失败，代码如下：

```
kafka-topics.sh --create --Zookeeper 10.200.5.88:2181,10.200.5.66:2181,10.200.
5.32:2181/kafka --replication-factor 2 --partitions 6 --topic accesslog-topic
```

最后展示下目前创建的 accesslog-topic：

```
kafka-topics.sh --describe --Zookeeper 10.200.5.88:2181,10.200.5.66:2181,10.200.
5.32:2181/kafka --topic accesslog-topic
```

```
Topic:accesslog-topic PartitionCount:6 ReplicationFactor:2 Configs:
  Topic: accesslog-topic Partition: 0 Leader: 3 Replicas: 3,2
Isr: 3,2
  Topic: accesslog-topic Partition: 1 Leader: 0 Replicas: 0,3
Isr: 0,3
  Topic: accesslog-topic Partition: 2 Leader: 2 Replicas: 2,0
Isr: 2,0
  Topic: accesslog-topic Partition: 3 Leader: 3 Replicas: 3,0
Isr: 3,0
  Topic: accesslog-topic Partition: 4 Leader: 0 Replicas: 0,2
Isr: 0,2
  Topic: accesslog-topic Partition: 5 Leader: 2 Replicas: 2,3
Isr: 2,3
```

为了验证消息集群是否成功，可以使用 Kafka 自带的 kafka-console-producer.sh 和 kafka-console-consumer.sh 进行消息发送和接收的测试。Kafka 虽然提供了对消息的存储，但是考虑到数据的规模，也可以将其中的消息写入 HDFS，这样在实时处理的同时，也会为将来的批量处理做好准备。Kafka 集群部署的更多细节请参考 Nishant Garg 所著的《Apache Kafka》。

这个方案的最后就是 Storm<sup>①</sup> 集群的部署。在每台 Storm 机器上，进入其 conf 目录并修改 storm.yaml 配置文件，代码如下：

```
storm.Zookeeper.servers:
- "10.200.5.88"
- "10.200.5.66"
- "10.200.5.32"
```

① 本书使用的 Storm 版本是 0.9.2-incubating。

```
nimbus.host: "10.200.5.88"
```

```
storm.zookeeper.port: 2181
```

```
supervisor.slots.ports:
```

- 6700
- 6701
- 6702
- 6703

```
storm.local.dir: "/Users/daobao.yang/bigdata/storm"
```

Storm 集群的主节点称为 Nimbus，从节点称为 Supervisor，我们需要在 10.200.5.88 上启动 Nimbus 服务：

```
nohup storm nimbus &
```

在其他两台机器上启动 Supervisor 服务：

```
nohup storm supervisor &
```

消息通过各种渠道进入 Kafka 消息中间件，然后再经由 Storm 进行流式处理，因此要在 Storm 的 Spout 中读取 Kafka 中的消息。好在 0.9.2-incubating 这个版本的 Storm 已经自带了一个 Kafka 集成插件，称为 storm-kafka，可供直接使用。之后，需要通过编程来自定义 Storm 的 Spout 和 Bolt 类，最终实现具体的处理逻辑。具体细节请参考 P.Taylor Goetz 等人所著的《Storm 分布式实时计算模式》。

设计完这些，大宝的团队将在线用户行为的跟踪、记录和处理体系基本上搭建出来了。在实践中他进一步确定，其实不存在最优秀的架构方案，要根据实际业务的需求，具体情况具体分析，找到最合适的路线。

## 这个搜索有点逊

自从大宝团队将自家的用户行为跟踪和采集系统上线后，顾客在其网站上的数据得到了有效的收集和分析。管理层每天都可以及时地看到报表，对决策大有帮助，技术部门也获得了更多的优质数据用于提升挖掘算法的效果，客户体验得到进一步提升，真可谓一举多得。不过，问题也随之而来，创业核心团队发现搜索的转化率相对于其他行业的竞争对手而言，明显处于一个低位，大约有 20% 的差距。于是，大家再次将目光聚焦到全站的搜索功能上，大宝和小丽分别作为技术和业务的带头人，坐在一起仔细分析这个问题。

“大宝，你知道的，我们的搜索虽然上线很久了，但是还有很多改进空间。”

“嗯，确实是，我最近也收到不少关于这项功能的反馈，主要是搜索的商品范围有限，没有办法搜索到促销商品和团购商品。同时，精准度也不是很好，搜索结果中经常会有不相关的商品排在前面。”

“看来你已经有所耳闻了，这两个确实是主要问题。还有几个小问题，我长话短说，业务的同事经常反馈在后台更新了商品发布的信息，但是前端访问的还是老数据，要很久才会更新；此外，前端搜索页面有时打开速度非常慢，用户没有耐心等待；最后，搜索下拉框也没有任何提示，很不方便。”

“看来问题还不少，”大宝挠了挠头，“不过你放心，这些对于我们技术部来说都不是事。”

“太好了，你办事我放心。接下来的几周，我们逐个过一下每个问题的细节。”小丽冲着大宝会心一笑，毕竟随着磨合的深入，彼此之间越来越有默契了。

## 12.1 业务需求：还要搜得更多

首先，最为重要的问题是解决搜索商品覆盖面少的情况。在和促销业务及团购业务的



部门负责人沟通后，大宝了解了他们的痛点。

促销的业务主要是实现各种形式的促销，帮助线下的店铺提升销售业绩。具体的形式包括满额减价、满件减价、满额送赠品、定额任选等，促销手段的丰富程度则令人咋舌，竟然有数十种之多。不过导致的结果就是，顾客在购买决策的时候陷入了选择障碍。由于不清楚哪些商品参加了哪些活动，用户很难弄清楚买哪些商品更经济实惠。因此，促销的业务人员希望能有一个搜索功能，允许用户查找参加某个促销活动的商品到底有哪些。当然，在促销活动中，根据分类和关键词再次缩小查询范围，也是更好的附加功能。

而团购的业务方，更为看重的是搜索带来的关键词查询和筛选能力。之前的团购是采用数据库 SQL 语言的查找来实现的。在业务的初期，团购主要通过不同的频道来实现，例如水产频道有阳澄湖大闸蟹团购，数码频道有 Apple iPad Pro 团购，时尚频道有爱马仕箱包团购等。用户只需要在限定的频道内浏览即可，SQL 查询也毫无压力。但是随着团购商品和用户访问量的增加，关键词搜索的需求被提到议事日程上。这时，SQL 语言模糊型的关键词匹配出现了性能瓶颈，而且越来越明显。此外，它也没法提供开源搜索系统中自带的切面 (Facet) 功能。

大宝仔细观察了一下这些数据，发现这些商品其实已经被全站的搜索引擎收录，只是促销商品还没有相关的促销信息，而团购商品也还没有相关的团购信息。经过一阵思考，他隐约觉得这些都可以融入同一个搜索引擎来实现。如果不能融合，那就意味着每接受一个新的商品搜索需求，可能就要建立一个全新的搜索集群，无论对于开发、部署还是维护，这都将是个灾难。此外，除了不同的业务形态，第三方内容的不一致性也极大地影响了数据的集成。因为是一个 O2O 平台，所以会有很多第三方的线下商铺加盟。而由于历史的原因，他们所使用的大都是不同的 ERP 系统，数据格式也大相径庭。为了能够接入大宝团队的平台，原有商铺的数据必须经过逐一转换。然而过于严苛的关系型数据库将会耗费技术和运营团队大量的时间和精力。考虑到这两大因素，当务之急是需要一个高效的融合方案，可以便捷地将不同的数据源集成起来，并塞入 Solr 的搜索引擎中。

## 12.2 “还要搜得更多”：产品设计和选型

为了满足数据集成的需求，大宝必须要修改 DIH 这种数据导入的方式。因为 DIH 通常用于将数据从关系型数据库导入到 Solr 中。虽然导入非常方便，但是关系型数据库对于数据定义 Schema 的要求非常严格，同一张表里的数据必须要有同样的字段。如果仍然使用 DIH，那么就可能意味着要使用如下两种做法：

第一种，在关系型数据库里设置一个宽表，所有的商品拥有所有的字段。那么，不是促销的商品必须要有促销相关的字段，不是团购的商品也需要有团购相关的字段，这势必会导致很多字段都是冗余和浪费的，会影响表数据更新的性能，不利于数据库的维护。

第二种，需要将商品基本信息、促销信息、团购信息放入到不同的表，然后进行连接

(Join) 操作，这无疑会加大 SQL 语句的复杂程度和执行时间。新的业务类型越多，数据种类就会越多，那么 DIH 的效率就会越差，不利于今后需求的扩展，也不利于搜索索引系统的维护。

最后，他想到了第 3 章介绍的 HBase。根据小明的建议来看，HBase 并没有严格的 Schema 定义，对于集成异构的数据源而言非常灵活和高效。在设计 HBase 的宽表之初，我们只需要指定列族 (Column Family) 即可，具体的列限定符 (Column Qualifier) 可以在需要的时候再添加。例如，最开始仅仅是处理普通商品，这时只需要设置商品名称、导购属性、卖家信息等列的限定符。如果哪天某个商品突然被选为团购商品，那么可以动态地加入团购价格、团购数量、开始和结束时间等列的限定符。如果该商品的团购结束了，则可以再次动态地删除这些团购信息，对其他商品完全没有影响。

基于这些特性，大宝团队提出了大致的架构，如图 12-1 所示。

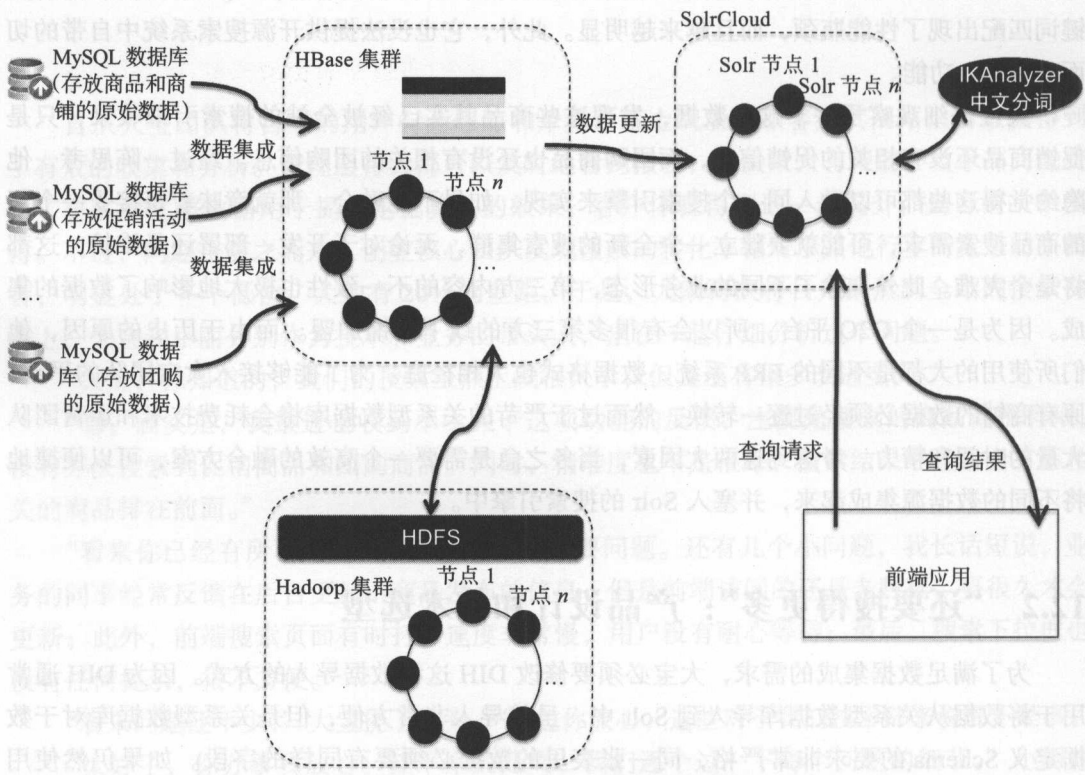


图 12-1 引入 HBase 模块的搜索架构

其中，DIH 模块被 HBase 集群取而代之，因此缺失了将数据直接导入 Solr 数据库中这个便捷的功能，这使得从多个关系数据库导入数据到 HBase，以及 Solr 读取 HBase 的数据进行索引这两个步骤都需要额外的编程。不过，换来的是灵活的异构数据源集成。另外，相

比 DIH，这种架构还有两个明显的优势，那就是提升了整体更新的速度，并且还可以利用 HBase 构建一层缓冲。

- 提升更新速度：DIH 中的 SQL 查询语句可能使用了连接（Join）操作，只要涉及的数据量一旦增大，执行势必会变得很缓慢，这就导致了最终更新流程变长，从时间上看甚至是指数级的增加。如果是采用 HBase，就可以分批次地将更新字段逐步写入 HBase 中，这样就能取消关系型数据库中的连接操作，大大提升效率。
- 构建数据缓冲：如果是 DIH 直接将数据导入 Solr 中，那么每当 MySQL 数据库中的数据发生变化，DIH 的增量更新就会修改 Solr 里的数据，当这个更新量达到足够大的规模时，则会导致 Solr 系统的频繁地磁盘写入操作，这一定会影响 Solr 系统的读取性能，最后使得前端应用的请求响应速度降低，增加高并发的可能性，严重的情况下甚至会造成系统的崩溃。若要在这种情况下进行优化，就需要增加 Solr 的集群节点，而这对于用户查询量不高的情况而言就是一种浪费。如果是采用 HBase，那么频繁的数据更新就会写入 HBase 中，只要我们控制好 Solr 同步 HBase 数据的节奏，那么最差的情况就是 HBase 系统崩溃，Solr 没有及时读取到最新的数据，但是仍然可以对前端提供正常的搜索服务，可以认为这是一种服务自动降级。只是这种情况下的优化，就要关注 HBase 集群的扩容了。

## 12.3 “还要搜得更多”的方案实现

### 12.3.1 HBase 的部署

第 11 章中简介了 Hadoop 集群的部署，在 Hadoop HDFS 及 Zookeeper 集群正常运行的基础上，这里简介下 HBase<sup>①</sup> 集群的搭建。进入 HBase 的 conf 目录，修改 hbase-site.xml，添加如下的内容：

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://10.200.5.88:9000/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>10.200.5.88,10.200.5.66,10.200.5.32</value>
  </property>
```

① 本书使用的 HBase 版本是 0.98.5。



```
<property>
  <name>hbase.master</name>
  <value>10.200.5.88:60000</value>
</property>
<property>
  <name>Zookeeper.session.timeout</name>
  <value>60000</value>
</property>
<property>
  <name>hbase.Zookeeper.property.clientPort</name>
  <value>2181</value>
</property>
<property>
  <name>hbase.Zookeeper.property.dataDir</name>
  <value>/Users/dabao.yang/bigdata/Zookeeper-3.4.6/data</value>
</property>
</configuration>
```

其中，hbase.rootdir 指定的 IP（或主机）和端口需要和 Hadoop 的配置一致，hbase.Zookeeper.quorum、hbase.Zookeeper.property.clientPort、hbase.Zookeeper.property.dataDir 和 Zookeeper 的配置保持一致。保存配置文件，然后使用 start-hbase.sh 启动 hbase，代码如下：

```
start-hbase.sh
localhost: starting Zookeeper, logging to /Users/dabao.yang/bigdata/hbase-0.98.5-hadoop1/bin/./logs/hbase-dabao.yang-Zookeeper-localhost.out
starting master, logging to /Users/dabao.yang/bigdata/hbase-0.98.5-hadoop1/logs/hbase-dabao.yang-master-localhost.out
localhost: starting regionserver, logging to /Users/dabao.yang/bigdata/hbase-0.98.5-hadoop1/bin/./logs/hbase-dabao.yang-regionserver-localhost.out
```

启动成功后，可以在 HDFS 文件系统中找到 hbase 的目录，如图 12-2 所示。这是依据 hbase-site.xml 的设置而来的。

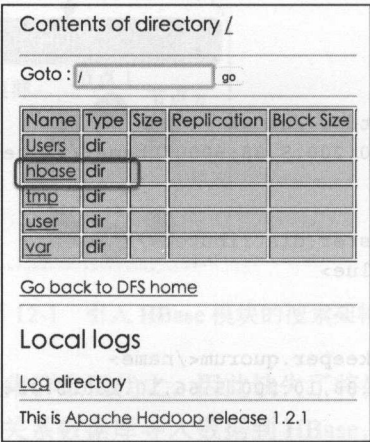


图 12-2 HDFS 中显示 hbase 目录创建成功



最后，使用 hbase shell 命令启动 HBase shell，测试 HBase 上基本的读/写功能。首先通过 list 来查看目前有哪些表，结果为空：

```
hbase(main):002:0> list
TABLE
0 row(s) in 0.0440 seconds
```

然后创建 1 张表格，代码如下：

```
hbase(main):003:0> create 'o2o_product', 'info'
0 row(s) in 0.5440 seconds
```

```
=> Hbase::Table - o2o_product
```

```
hbase(main):004:0> list
```

```
TABLE
o2o_product
1 row(s) in 0.0310 seconds
```

```
=> ["o2o_product"]
```

```
hbase(main):005:0> describe 'o2o_product'
```

```
DESCRIPTION      ENABLED
```

```
'o2o_product', {NAME => 'info', DATA_BLOCK_ENCODING => 'NONE', true
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1'
, COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER',
KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY
=> 'false', BLOCKCACHE => 'true'}
```

```
1 row(s) in 0.0470 seconds
```

```
hbase(main):006:0>
```

我们发现，HBase 中创建表格的 create 语法非常简单，由于没有严格的 Schema 定义，因此只需要提供表名 o2o\_product 和列族 info 即可，无须提供具体的字段名和类型，而 describe 命令会显示表格的基本信息。接下来，可以通过 put 命令来插入第 1 个商品的数据，代码如下：

```
hbase(main):029:0> put 'o2o_product', 'product1', 'info:product_name',
'Imported Milk Package'
```

```
0 row(s) in 0.0080 seconds
```

```
hbase(main):031:0> put 'o2o_product', 'product1', 'info:product_price',
'188.88'
```

```
0 row(s) in 0.0060 seconds
```

其中 product1 相当于商品的 ID，product\_name 和 product\_price 分别是商品的名称和价格。然后插入第 2 个商品的数据，不过第 2 个商品是团购商品，比第 1 个商品多出了团购价格 group\_price、团购开始时间 group\_start 和结束时间 group\_end，代码如下：

```
hbase(main):034:0> put 'o2o_product', 'product2', 'info:product_name', 'A New Movie'
0 row(s) in 0.0050 seconds
```

```

hbase(main):035:0> put 'o2o_product', 'product2', 'info:price', '30.00'
0 row(s) in 0.0050 seconds

hbase(main):036:0> put 'o2o_product', 'product2', 'info:group_price', '10.00'
0 row(s) in 0.0060 seconds

hbase(main):037:0> put 'o2o_product', 'product2', 'info:group_start',
'2016/01/08'
0 row(s) in 0.0070 seconds

hbase(main):038:0> put 'o2o_product', 'product2', 'info:group_end',
'2016/01/15'

```

最后，用 scan 命令查看插入的信息，代码如下：

```

hbase(main):039:0> scan 'o2o_product'
ROW COLUMN+CELL
  product1 column=info:product_name, timestamp=1448417509669, value=Imported
Milk Package
  product1 column=info:product_price, timestamp=1448417598308, value=188.88
  product2 column=info:group_end, timestamp=1448417745410, value=2016/01/15
  product2 column=info:group_price, timestamp=1448417716731, value=10.00
  product2 column=info:group_start, timestamp=1448417733315, value=2016/01/08
  product2 column=info:price, timestamp=1448417699278, value=30.00
  product2 column=info:product_name, timestamp=1448417685652, value=A New Movie
2 row(s) in 0.0400 seconds

```

可以看出 product1 和 product2 都已经存储在 HBase 之中。不过，每一行都是某条记录中的某个字段。这点和关系型数据库中每一行就是一条完整的记录有所不同。

### 12.3.2 HBase 和 Solr 的集成

因为缺乏 DIH 模块的支持，所以开发者需要编码完成如下两部分的内容：

- ❑ 将全量和增量的数据更新从关系型数据库导入到 HBase 中。当然，如果业务逻辑并不复杂，也可以直接利用 Apache Sqoop 这样的工具。
- ❑ 利用 HBase 的 scan 命令和时间戳，获取全量和增量的数据更新，并写入 Solr 集群中。

Solr 有个很好的特性，就是动态字段（Dynamic Field），它也提供了灵活的 Schema，能和 HBase 中的数据对应起来。例如，上节测试样例中第 2 个商品的团购的相关信息，并不是每个商品都必需有的，因此我们可以修改 Solr 的 schema.xml，加入如下字段定义：

```
<dynamicField name="group_*" type="string" indexed="true" stored="true"/>
```

这样，如果某个商品是团购商品，就可以在添加 Solr 文档时，提供 group\_ 前缀开头的字段，否则就没有必要提供。

## 12.4 业务需求：还要搜得更准

近期用户时常抱怨一个问题，那就是关键词搜索的结果非常不精准。搜索“牛奶”，很多牛奶巧克力，甚至是牛奶色的连衣裙都跑到搜索结果的前排了，用户体验非常差。但是，巧克力和连衣裙这种商品标题里确实存在“牛奶”的字样，如果简单地抹去，又会导致搜索“牛奶巧克力”或“牛奶色连衣裙”时无法展示相关的商品，这肯定也是用户无法接受的。但搜索不精确的情况却又十分普遍，例如，搜索“橄榄油”的时候会返回热门的“橄榄油发膜”，或者是“橄榄油护手霜”；搜索“手机”的时候会返回热门的“手机壳”和“手机贴膜”，这种例子不胜枚举，加上商品的品类也在持续增加，因此也无法完全通过人工运营来解决这个问题，图 12-3 列举了“橄榄油”的案例，左边是现状，右边是用户的期望，差距非常明显。



图 12-3 左右相比，相关性有明显的差距

那该如何更精准地返回搜词结果，将更为相关的商品排在前列呢？这一直是大宝挥之不去的痛，但是一时间他也不知道该如何解决。只能再次请黄小明出马了。大宝将困境一五一十地告诉了小明。

“哈哈，你算问对人了。我最近在专门研究这个课题，根据目前线上测试的结果来看，非常有效，我这里跟你共享下其核心的技术思想。”

“太感谢了！”



“对了，你们有没有记录用户的行为？这对于精准性问题的解决是非常关键的。”

“已经上线了，用户行为的数据目前都有完整的收集和存储。”

“嗯，那就好办多了。”

## 12.5 “还要搜得更准”：产品设计和技术选型

### 12.5.1 提升搜索排序的相关性

小明首先向大宝说明了为什么会产生搜索结果不精准的情况，主要是 Lucene 默认打分机制惹的祸。第 5 章信息检索的搜索部分已经介绍过，Solr 的底层是使用 Lucene 来实现的，因此也继承了 Lucene 的相关性评分衣钵，主要的模型所考虑的都是关键词的 tf-idf、文章的长短、查询的长短等因素。这种方式非常适合普通文本的检索，在各大通用搜索引擎里也被证明是行之有效的办法之一。但是，该方法明显不适合电子商务的搜索平台，主要原因有：

- ❑ 商品的标题都非常短。目前电子商务网站的搜索引擎通常只对商品的标题或名称进行索引，而不会针对商品的具体描述来进行索引。这主要是因为描述里面的内容过于丰富，还包括不少广告宣传，不一定是针对产品特性的信息，如果对描述进行索引，不仅加大了系统计算和存储的负担，还会导致较为低下的精确度。这就导致了商品的标题、名称和主要的导购属性成为了搜索索引关注的对象，而这些内容一般都短小精悍，不需要考虑其长短对于相关性衡量的影响。
- ❑ 关键词出现的位置、词频对相关性的意义不大。如上所述，正是由于商品搜索主要关注的是标题等信息浓缩的字段，因此某个关键词出现的位置、频率对于相关性的衡量影响非常有限。如果考虑了这些，反而很容易被别有用心卖家所利用，进行不合理的关键词搜索优化（SEO），导致最终结果的质量变差。
- ❑ 用户的查询普遍比较短。普通搜索引擎中有海量的网页信息供用户查阅，因此用户为了准确地找到其所寻找的信息，可能会输入尽可能多的关键词。然而，在电商平台上，商品的数量相对于互联网的网页信息量而言可谓冰山一角，顾客无需太多的关键词就能定位大概所需，因此查询的字数多少对于相关性衡量也没有太大的意义。

因此，电商的搜索系统不能局限于关键词的词频、出现的位置等基础特征上，而应该从其他方面来考虑。前面大宝所犯愁的问题，实际上主要纠结在一个“分类”的问题上。例如，顾客搜索“牛奶”字眼的时候，系统需要弄清楚用户是期望找到饮用的牛奶，包括鲜奶、包装奶、进口奶、酸奶等，还是牛奶味的巧克力或饼干。

“那如何才能得知这样的信息呢？人是很容易理解，但是系统没法得知啊，难道要人为地逐个运营吗？顾客的输入千变万化，而且商品的品类也会逐步更新，人工设置的工作量实在是太巨大了。”

“大宝，别急，我们完全可以利用用户的行为来做判断。”



“哦？具体怎样操作呢？”

“其实主要思路并不复杂，就是观察用户在搜词后的行为，包括点击进入的详情页、是否添加到购物车等，这样我们就能知道对于每个关键词而言，顾客最为关心的是哪些类目标了。”

“哇，这真是个好主意！我之前怎么没有想到呢。”

“嗯，所以说这个问题没有想象的那么难。举个例子，当用户输入关键词‘咖啡’，如果他经常浏览和购买的品类是国产冲饮咖啡、进口冲饮咖啡和咖啡饮料，那么这 3 个分类就应该排在更前面，然后将其他虽然包含了咖啡字眼，但是并不太相关的分类统统排在后面。同时，这个方法在一定程度上还能解决季节性的问题，到了夏季，人们查找咖啡时更希望喝到冰爽的饮料，而不是冲饮。到了冬季，人们更多的是希望找到热乎乎的冲饮，而不是饮料，这些排序需求都可以通过用户的行为来调整。基于这些，系统的架构基本上就是这样设计的。”小明画出了一个框架（如图 12-4 所示）。其中，搜索引擎部分仍然采用上一节大宝设计的方案，通过 HBase 融合异构数据源，然后用 Solr 通过 HBase 建立整体的商品索引，并向前台提供搜索服务。而数据收集和分析仍然采用第 11 章的设计方案，通过 Flume 和 Kafka 分别收集批量数据和实时数据，通过 Hive 和 Storm 分别进行数据的批量处理和实时处理，最后将用户输入的关键词和期望分类的直接关系，输入到 Solr 进行重新排序，从而提升相关性。

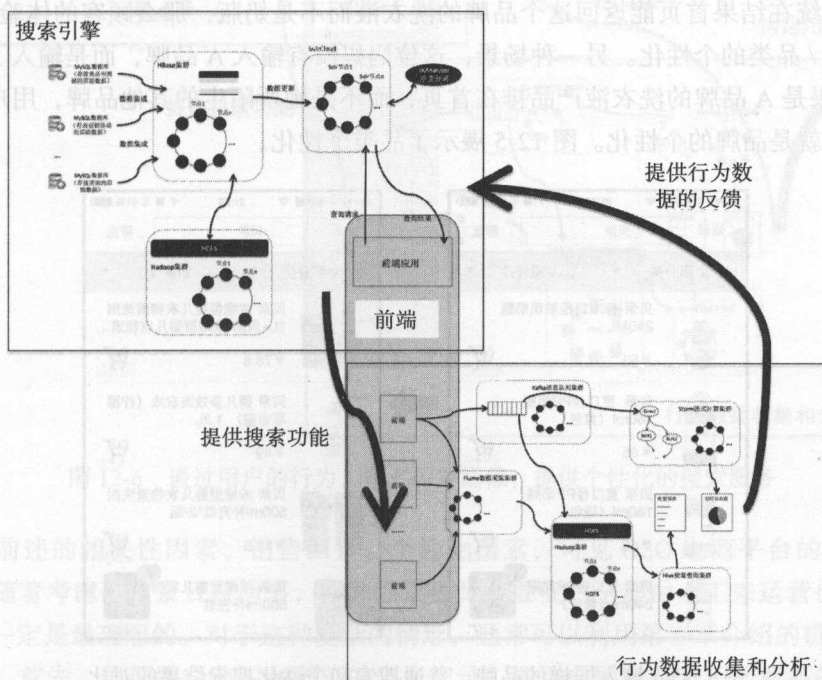


图 12-4 通过用户的行为，提升搜索引擎的精准度

## 12.5.2 提升搜索排序的整体效果

大宝和小明就这个问题的细节又进行了一些探讨，没想到这下小明的话匣子一下子被打开了，除了相关性，他还聊了搜索排序的其他很多话题。

除了搜索关键词的相关性是个重要的方面外，在电商平台上，类目和关键词搜索还需要经常考虑的问题就是商品排序的合理性。需要经常考虑的因素包括不同周期的销量和销售额、不同地域的销量和销售额、是否市场新品、是否平台新品、是否价格促销、是否季节品、商家服务质量等，这里简单地概括为销售因素。这些都需要从不同的数据源中获得，HBase 再一次体现其集成异构数据的强大能力。同时，这些数据通常对实时性的要求都不高<sup>①</sup>，因此，可以将这些信息建入 Solr 的索引，利用静态分值进行排序，从而减少动态排序这种对系统性能消耗较高的操作。

对于高级的搜索应用而言，个性化也是非常重要的。人们容易产生一个误区，那就是认为只有推荐才需要个性化，其实不然，搜索同样也需要。第 5 章曾简单对比过搜索和推荐，搜索的输入是明确的关键词，而推荐往往没有明确的查询条件。所以，搜索要求的是精准，而推荐在某种程度上需要满足的是用户对“新颖”的渴望，从这个角度而言，搜索更需要准确的个性化。假想一下，A 品牌的奶瓶在全网是非常畅销的，大部分的顾客在搜索 A 品牌时都会选择相应的奶瓶产品。此时，一位 5 岁儿子的妈妈在网站上进行关键词搜索，虽然她的儿子早已过了用奶瓶的阶段，不过她一直在购买 A 品牌的儿童洗衣液，如果她输入 A 品牌后系统在结果首页能返回这个品牌的洗衣液而不是奶瓶，那么顾客的体验就会更佳，这就是分类 / 品类的个性化。另一种场景，这位妈妈没有输入 A 品牌，而是输入了“儿童洗衣液”，如果是 A 品牌的洗衣液产品排在首页，而不是她所陌生的其他品牌，用户的体验也会更佳，这就是品牌的个性化。图 12-5 展示了品类个性化。



图 12-5 输入同样的品牌，普通搜索和个性化搜索结果的对比

① 除了季节性变化等。

在这些业务场景下,用户行为数据尤为重要,系统可以获取并分析用户对品牌、分类、价位等因素的喜好,然后利用这些数据建立完善的用户画像,最终,用户画像为搜索或推荐提供每位用户的细化数据,使得个性化服务成为可能。其整体架构如图 12-6 所示。与图 12-4 相比,图 12-6 主要多了用户画像的构建和使用模块。用户画像可能会根据应用场景的不同而有很多不同的划分方式,而且随着市场扩张,其规模也会日益庞大,因此这里也选择列式存储 HBase,以达到异构数据整合和横向水平扩展的目的。考虑到搜索的查询是在线的实时性服务,因此在 HBase 的基础之上加入 Redis 作为缓存,用于大幅提升画像数据的读取性能。

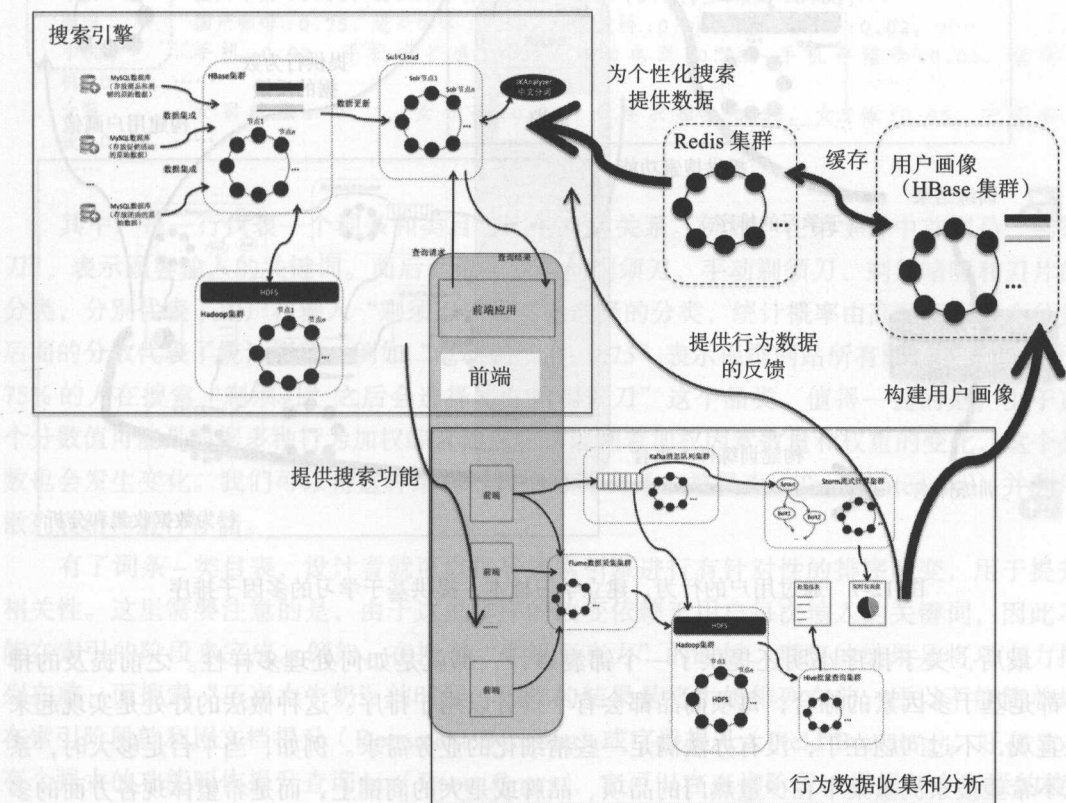


图 12-6 通过用户的行为,建立用户画像,提供个性化的搜索服务

综合前述的相关性因素、销售因素、个性化因素,可见 O2O 电商平台的搜索并不简单。那么随着考虑的因素日益增多,哪种因素更为重要呢?全部由人工来运营也不太现实,效果也不一定是最理想的。对于这种复杂的情形,通常可以利用第 6 章介绍的机器学习的方式来解决。首先,根据用户购买和浏览的历史记录,甄别出顾客对于不同商品的喜爱程度,并将其作为训练的样本,记录其所有因素的值。然后采用回归技术,确定对于理想的物品,



每个因素应该有怎样的权重取值。最终，这些取值将确定在未来排序时，不同因素对决策的贡献程度。整体架构图可以参考图 12-7。

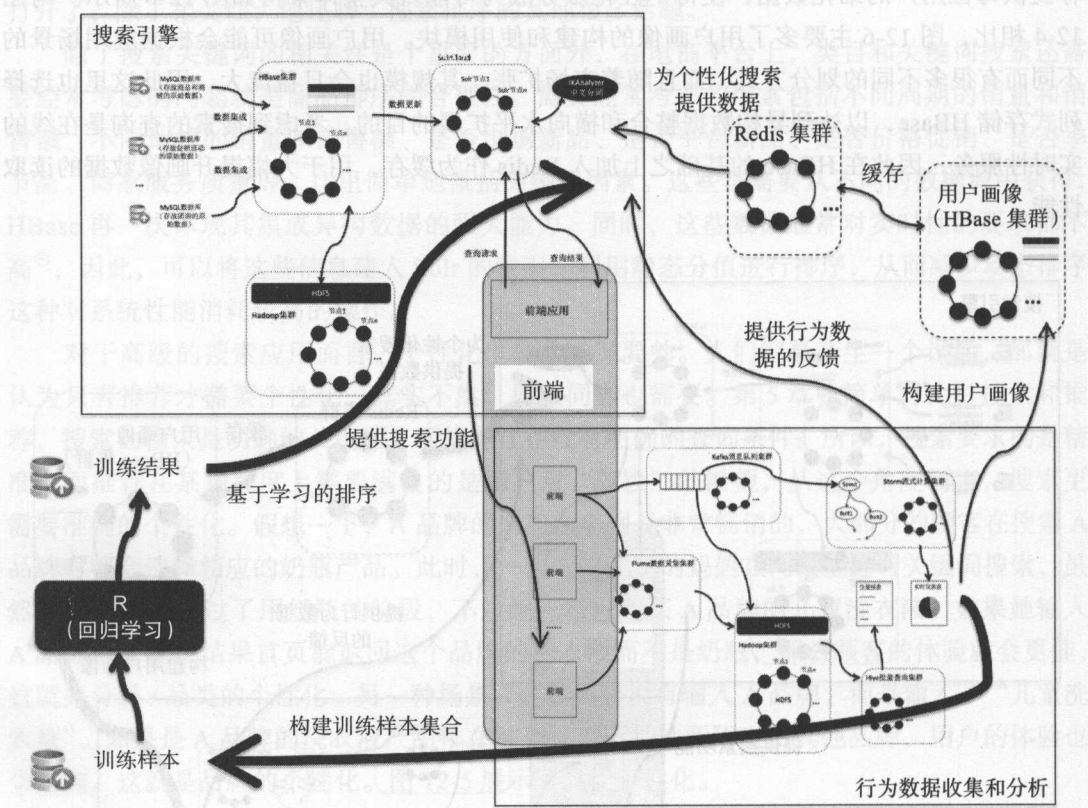


图 12-7 通过用户的行为，建立学习样本，提供基于学习的多因子排序

最后，关于排序小明还提供了一个锦囊妙计，那就是如何处理多样性。之前提及的排序都是基于多因素的综合，每项商品都会有个得分以用于排序。这种做法的好处是实现起来很直观，不过问题在于，没有办法满足一些精细化的业务需求。例如，当平台足够大时，系统不希望搜索流量集中在少量热门的品项、品牌或是大的商铺上，而是希望体现各方面的多样性，展示平台的丰富度。举个例子，当顾客搜索“剃须刀”的时候，业务和运营可能期望在搜索结果的前排既出现电动剃须刀，又出现手动剃须刀；既出现大卖场的货，又出现精品生活馆的货；既出现 A 品牌，也出现 B、C 和 D 品牌。这个时候，普通搜索引擎的实现方式（无论是 Lucene，还是 Solr 或 Elasticsearch）对于这个需求都是无法实现的。必须要在这些开源引擎的返回结果上再次加工，进行二次甚至是多次排序。不过，这个操作对于内存和 CPU 的消耗较高，容易导致搜索请求响应变慢，在高并发的情况下甚至会导致宕机和崩溃，因此需要有足够技术背景和经验的人来主导开发。



## 12.6 “还要搜得更准”的方案实现

为了弄清楚用户在搜索某个关键词时，对哪类商品更感兴趣，我们首先要定义观察用户的哪些行为，常见的行为特征包括购买和浏览。第 11 章已经展示了 Hive 的强大功能，它可以从行为日志中获取相关信息，在输入某个关键词后，根据用户经常购买了或浏览了哪些品类，进行一个占比的统计。对于购买和浏览两种行为，可以进行加权平均，通常购买的权重更高一些。这样，我们可以获得类似如下数据：

搜索词	相关类目列表
剃须刀	电动剃须刀 : 0.75, 手动剃须刀 : 0.19, 剃须啫喱 : 0.04, 刀片 : 0.01, .....
牛奶	国产牛奶 : 0.58, 进口纯牛奶 : 0.19, 酸奶 : 0.11, 风味奶 : 0.04, .....
咖啡	国产咖啡 : 0.75, 进口咖啡 : 0.15, 咖啡饮料 : 0.04, 进口咖啡豆 : 0.02, .....
手机	手机 : 0.92, 手机保护套 : 0.02, 移动电源 : 0.01, 手机存储卡 : 0.01, 蓝牙耳机 : 0.01, .....
女装	女羽绒服 : 0.31, 女外套 : 0.23, 女连衣裙 : 0.15, 女T恤 : 0.05, 中老年女装 : 0.03, .....
.....	

其中，每一行代表一个词条和类目之间的对应关系。例如，在第 1 行中首列是“剃须刀”，表示顾客输入的关键词。而后面依次是电动剃须刀、手动剃须刀、剃须啫喱和刀片的分类，分别代表了用户在输入“剃须刀”后可能选择的分类，统计概率由高到低。每个分类后面的分数代表了统计概率，例如“电动剃须刀: 0.75”表示根据网站所有顾客行为的统计，75% 的人在搜索“剃须刀”之后会选择“电动剃须刀”这个品类。值得一提的是，由于这个分数值可能是根据多种行为加权综合而来，所以随着加权因素数量和权重的变化，这个分数也会发生变化。我们可以将这种结构的数据称为词条-类目表 (Term-Category)，并利用散列表结构进行存储。

有了词条-类目表，设计者就可以指导搜索引擎进行有针对性的排序改变，用于提升相关性。这里需要注意的是，由于这里排序的改变依赖于用户每次输入的关键词，因此不能在索引的阶段来完成。例如，在搜索“牛奶巧克力”的时候，理想的结果是将巧克力排列在前，而搜索“巧克力牛奶”的时候，理想的结果是将牛奶排列在前，所以不能简单地在索引阶段就利用文档提升 (Document Boosting) 或字段提升 (Field Boosting)。好在 Solr 有个强大的功能叫作提升查询 bq (Boost Query)，它可以在查询阶段，根据某个字段的值，动态地修改命中结果的得分。因此，完全可以通过词条-类目表中的数据，构建包含 bq 参数的查询条件。仍然沿用第 9 章的例子，假设用户想搜索一个套餐，通过条件 `product_name:Package` 来查询，结果如图 12-8 所示。

排在第一位的是超市售卖的牛奶组合套餐，主要原因是牛奶商品的标题只包含 3 个单词，而火锅商品的标题包含了 4 个单词，默认的打分机制倾向于长度更短的结果。然而，词条-类目表的数据显示，多数用户在搜索“套餐”的时候，实际上更多地是想找餐馆，占到了 70% 的概率，超市快消品只占到了 10% 的概率，那么可以通过设置 bq 来进行修正。

如图 12-9 所示，点击 edismax 选项，将参数 bq 的内容设为 category\_name:(Food^0.7)OR category\_name:(FMCG^0.1)<sup>②</sup>，就可使得韩式火锅套餐排到了前列，更符合用户的预期，这完全是在没有修改索引的前提下实现的。使用 Solr bq 的更多细节，请参看 Trey Grainger 及 Timothy Potter 合著的《Solr in Action》这本书。

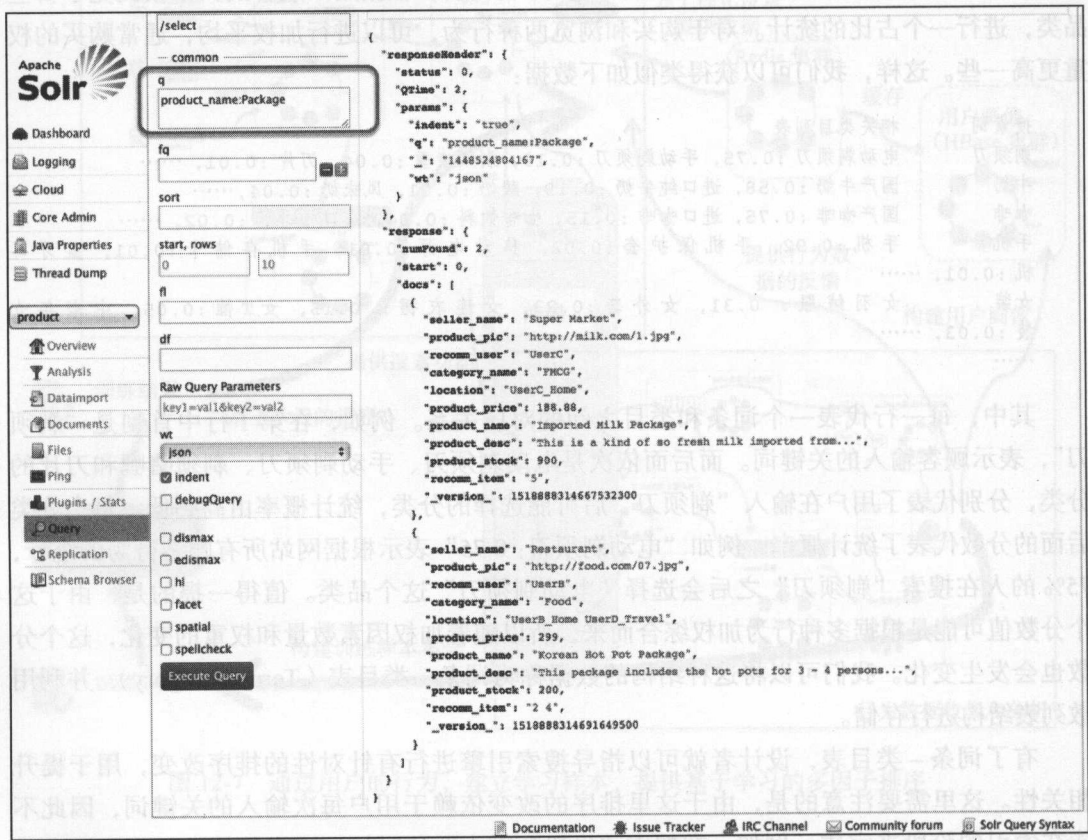


图 12-8 用户搜索关键词“套餐 (Package)”

这样，只要用户搜索行为的数据量足够大，质量足够高，那么完全可以在不进行大规模的人工干预的情况下，保证关键词搜索的相关性。以此类推，这个技术还可以用于个性化的排序。当用户登录后，我们可以根据顾客账号 ID 从用户画像中获取他 / 她对于不同品类、不同品牌的喜好程度，然后通过 bq 查询对不同的品类或品牌进行动态调整，实现类似图 12-5 所示的效果。

<sup>②</sup> 这里仍然使用了词条 - 类目表里的 0.7 和 0.1 的取值。实际中，根据原有打分，可能需要等比放大或缩小。

The screenshot shows the Apache Solr Admin interface. On the left, the 'Query' section is selected. The main area displays the 'Raw Query Parameters' and the 'edismax' query type. The 'bq' (Boosting) section is highlighted, showing the query 'category\_name:(Food^0.7) OR cate'. The right pane shows the JSON response with two items: a 'Restaurant' and a 'Super Market'.

```

{
  "response": {
    "numFound": 2,
    "start": 0,
    "docs": [
      {
        "seller_name": "Restaurant",
        "product_pic": "http://food.com/07.jpg",
        "recomm_user": "UserB",
        "category_name": "Food",
        "location": "UserB_Home UserD_Travel",
        "product_price": 299,
        "product_name": "Korean Hot Pot Package",
        "product_desc": "This package includes the hot pots for 3 - 4 persons...",
        "product_stock": 200,
        "recomm_item": "2 4",
        "_version_": 1518888314691649500
      },
      {
        "seller_name": "Super Market",
        "product_pic": "http://milk.com/1.jpg",
        "recomm_user": "UserC",
        "category_name": "FMCG",
        "location": "UserC_Home",
        "product_price": 188.88,
        "product_name": "Imported Milk Package",
        "product_desc": "This is a kind of so fresh milk imported from...",
        "product_stock": 900,
        "recomm_item": "5",
        "_version_": 1518888314667532300
      }
    ]
  }
}

```

图 12-9 用户搜索关键词“套餐 (Package)”，结果根据词条 - 类目数据进行动态调整

## 12.7 业务需求：还要更快

在解决了商品的覆盖面和相关性问题后，搜索体验有了明显的改善，转化率提升了 10% 以上，大宝总算长舒了一口气。不过，他还记得小丽反馈过两个“慢”的问题。第一个“慢”是指数据更新慢，业务人员修改了商品标题、类别或详细描述后，要经过数十分钟才能生效。第二个“慢”是指查询的响应速度，随着线下业务的快速扩张，网站的商品量和访问量都呈现明显地增长，搜索系统的负荷越来越大，每次查询的时间由 100 毫秒<sup>①</sup>左右增加到 800 毫秒左右，前端用户已经可以明显地感觉到搜索变慢。这两点都影响了业务规模的

① 1 秒等于 1000 毫秒。



进一步提升。由于大宝在此方面确实缺乏实战经验，因此他再次找到小明来帮忙。

## 12.8 还要“变”得更快：产品设计和技术选型

“后台进行更新操作后，搜索前台不能及时体现这个变化？那应该就是更新的机制有问题。”小明听过陈述后，果敢断定了第一个问题的症结，并替大宝梳理了一下目前搜索引擎的架构。从图 12-1 的架构中可以看出，HBase 虽然提供了更便捷的数据整合功能，但是无疑增加了数据处理的步骤，延长了数据更新的流程，所以实时性会变得更差。即使有增量更新的方式，也只能是定时地从关系型数据库扫描获取有变化的部分，并将其更新到 HBase，然后再从 HBase 扫描获取有变化的部分，最后批量写入 Solr Cloud 集群。为了提升及时性，只能加快扫描关系型数据库和扫描 HBase 的频率。可是，过于频繁的扫描会降低关系型数据库和 HBase 的读写性能，造成系统的隐患。

借鉴第 4 章探讨的实时性处理，采用 Kafka 这样的消息机制可以较好地解决问题。首先，Kafka 可以标记发生变化的数据记录，然后进行如下三种方式的处理：

- Kafka 从关系型数据库中获取到更新后的数据，直接将变化的内容推送到 Solr Cloud 集群，让其更新。这种处理方式的优势在于及时性最强，而且不会对关系型数据库和 HBase 产生扫表的压力。其劣势在于放弃了 HBase 这个数据层的缓冲，如果增量更新的数量庞大，那么很可能造成 Solr 的集群忙于索引的写入，从而导致前台查询响应变慢，甚至是发生搜索系统的宕机。因此，从整体而言，这种方式非常适合数据规模小，但是及时性要求很高的增量更新。
- Kafka 从关系型数据库中获取到更新后的数据，仅仅将变化的内容推送到 HBase 集群。而 Solr 从 HBase 更新数据，仍然是通过定期扫描的形式来实现的。相对于第 1 种方式，其优势在于不会对 Solr 集群产生过多的压力，能保证线上前台对搜索的访问，同时也降低了对关系型数据库的扫表压力。当然，其劣势就体现在及时性不强，只是加速了从关系数据库到 HBase 的数据更新，而没有加速从 HBase 到 Solr 的数据更新。这种方式更适合数据规模大、及时性要求不高的增量更新。
- Kafka 从关系型数据库中获取到更新后的数据，一方面将变化的内容推送到 HBase 集群，另一方面根据一定的阈值，来通知 Solr 进行更新。Solr 更新的速度可以通过时间间隔和更新数量的阈值来进行控制，以确保其读写的稳定性。可以认为该方式是前两种的折中方案。既避免了对关系型数据库和 HBase 的扫表，同时对实时性也有一定程度的保证，比较适合增量更新数量波动较大的场景。不过，其劣势在于实现方案更复杂，开发和维护的成本更高一些。

“对了，大宝，除了这三种更新方式，还需要注意推送到 Solr 的数据内容是如何设计的。一种是每次更新时都获取到完整的数据记录并将它们推送到 Solr，还有一种是只获取变化的字段并将其推送到 Solr。我个人认为前一种更好一些。”



“为什么呢？只关注变化的部分不是更好吗？不仅传输数据量少，而且 Solr 更新也快。”

“没错，虽然目前 Solr 的原子更新可以支持字段级的内容修改，但是不建议使用，主要原因是 Solr 的原子更新，其过程仍然是将原有的整条记录读取出来，然后再将更新部分替换掉，再次插回整条记录。这样就会增加 Solr 读的开销，造成线上服务的压力。因此，还不如将整条记录的内容直接对 Solr 进行写入。”

“哦，如果是这样，那每次都要读取全部的信息了，那有点麻烦啊。”

“确实，完整的数据可以从 HBase 里获取，或者是通过 Kafka 从关系型数据库里读取。是要求 Solr 的稳定，还是要求实现方案的简洁，就根据你们的实际情况来衡量吧。”

说完，小明画出了如图 12-10 所示的大体架构图。

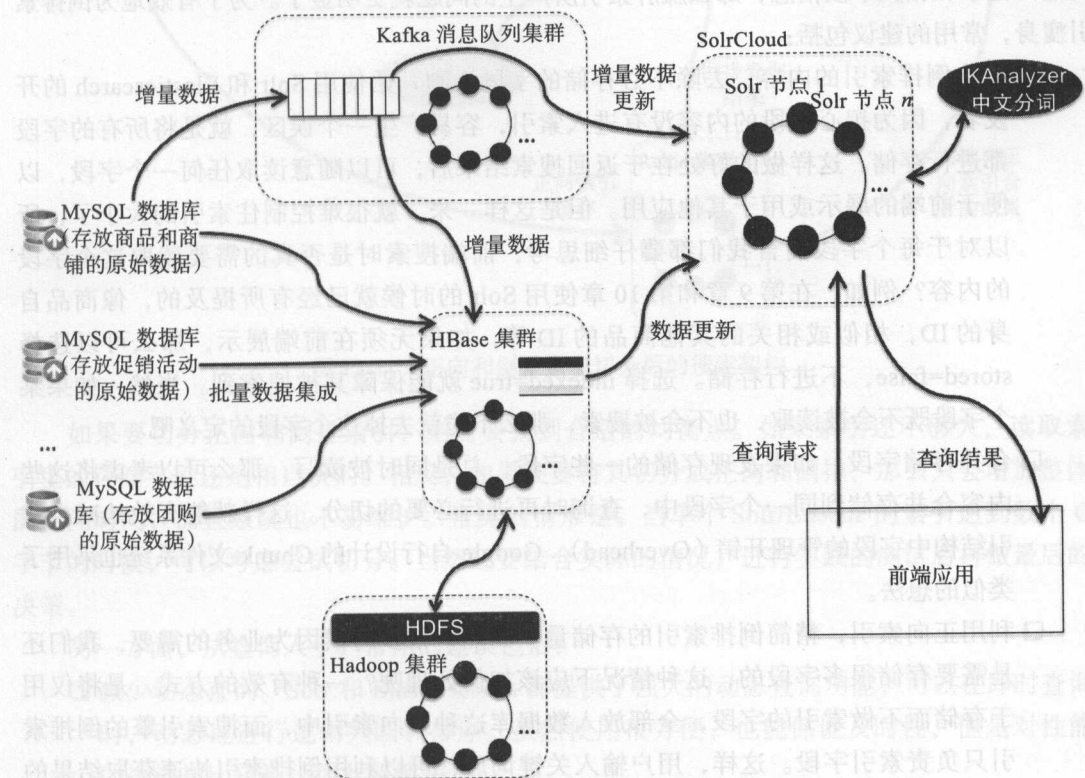


图 12-10 加入实时索引更新的搜索架构

## 12.9 还要“搜”得更快：产品设计和选型

“至于搜索请求变慢的情况，主要是有些设计的理念和方式不恰当。当商品数量和用户访问量还很小的时候，问题也许不会很明显。但是随着业务量的逐步增大，原来不是问题，

或者只是小问题的就可能演变为严重的问题。”在这方面小明也颇有心得，他指出，搜索引擎通常主要包含离线索引和在线查询两大模块，因此单台服务器的优化也要从这两大方面入手。

先来看索引这方面，为什么要简化索引，以及我们能做些什么。从第5章可以获知，倒排索引及其延伸的数据结构，对于搜索这样的信息检索系统而言是至关重要的。倒排索引会将数据对象包含关键词的关系转变为关键词到对象本身的关系，大幅提升了查询的效率。即便如此，设计者也不应该放任索引的规模进行不必要的增长。因为过大的索引，是不可能完全放入内存进行缓存的，一定会增加对磁盘的访问，那么较慢的磁盘 I/O 读写就会成为瓶颈，最终会导致查询变慢。特别是对于多次排序这种高级技术，往往需要从索引中读取更多的返回记录和相关字段信息，那么臃肿索引所产生的问题就更明显了。为了有效地为倒排索引瘦身，常用的建议包括：

- ❑ 简化倒排索引的内容，去除不必存储的字段。刚开始使用 Solr 和 Elasticsearch 的开发者，因为担心字段的内容没有进入索引，容易产生一个误区，就是将所有的字段都进行存储。这样做的好处在于返回搜索结果后，可以随意读取任何一个字段，以便于前端的展示或用于其他应用。但是这样一来，就很难控制住索引的大小了，所以对于每个字段而言我们都要仔细思考，前端搜索时是否真的需要读取这个字段的内容？例如，在第9章和第10章使用 Solr 的时候就已经有所提及的，像商品自身的 ID，相似或相关的其他商品的 ID 等，如果无须在前端展示，那么可以选择 `stored=false`，不进行存储。选择 `indexed=true` 就能保障其被搜索到。当然，如果某个字段既不会被读取，也不会被搜索，那么干脆就去掉这个字段的定义吧。
- ❑ 合并存储字段。如果发现存储的一些字段，总是同时被读写，那么可以考虑将这些内容合并存储到同一个字段中，查询时再进行必要的切分。这样就能有效地减少索引结构中字段的管理开销（Overhead）。Google 自行设计的 Chunk 文件系统也采用了类似的想法。
- ❑ 利用正向索引，精简倒排索引的存储量。当然，很多时候因为业务的需要，我们还是需要存储很多字段的，这种情况下应该如何处理呢？一种有效的方式，是将仅用于存储而不做索引的字段，全部放入数据库这种正向索引中，而搜索引擎的倒排索引只负责索引字段。这样，用户输入关键词后，可以利用倒排索引快速获取结果的 ID，然后再根据结果的 ID 到正向索引里获取其他存储内容。对于这种方案，需要注意的是优化正向索引读取的速度，如果正向索引读取过慢，那么无疑会拖累整体查询的速度，适得其反。此时，通常会利用 Redis 这种机制来做数据库的缓存，或者是直接构建 SOA 的服务模块<sup>①</sup>，提供高效访问正向索引的服务。大致的架构如图 12-11 所示。

① SOA 的概念和实现不是本书的重点，感兴趣的读者可以参考系统架构设计相关的书籍。

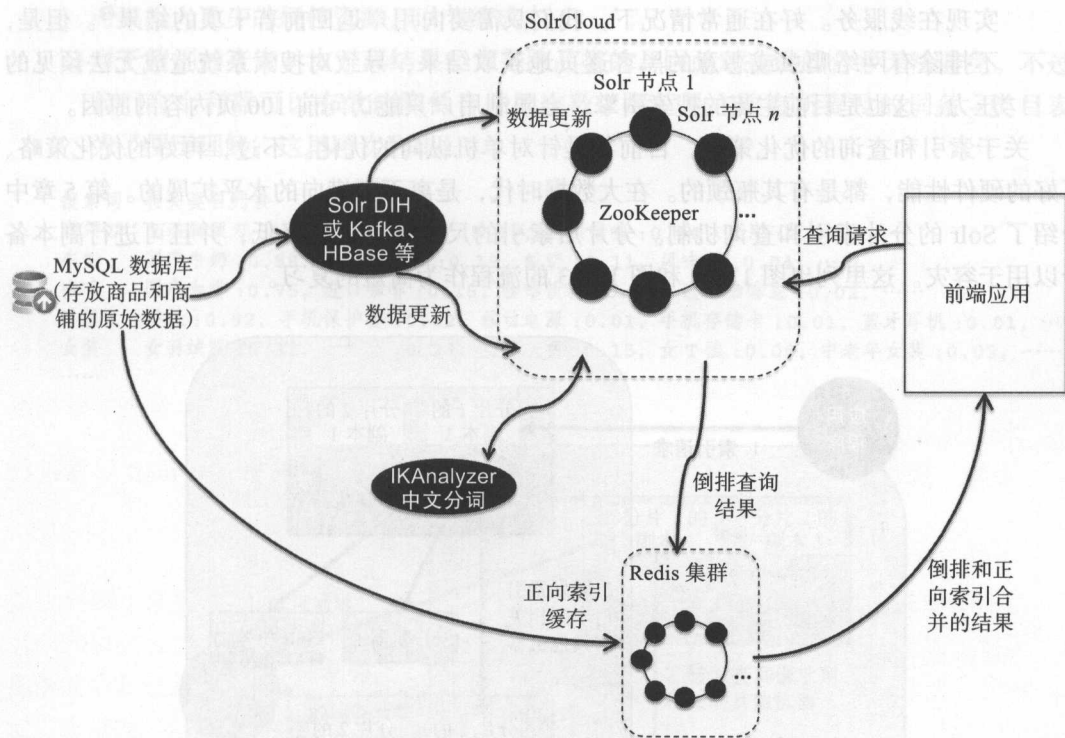


图 12-11 正向和倒排索引切分后的搜索架构

如果要切分正向和倒排索引，则需要找到合适的均衡点。如果索引还不够大，读取索引里的存储字段还是相当快的。相反，如果硬要将其切分成正向和倒排，那么只会增加整体的访问时间，而且系统也不易维护。常见的情形是，当单个 Solr/Lucene 的索引达到数十 G 字节的时候，可以考虑尝试切分。当然还要结合实际的情况，进行实践的测试后再做最后的决策。

另一方面，从查询入手，常用的建议包括：

- ❑ 减少动态查询。Solr 和 Elasticsearch 都提供了强大的动态查询功能，可以在即时查询时，动态地进行逻辑判断和排序。虽然使用很方便，也能保证及时性，但是对性能的损耗却比较高，因此要慎用。
- ❑ 尽量使用过滤查询（Filter Query）。前面在探讨相关性的时候，曾提到普通文本搜索引擎的打分机制并不一定适合所有的应用场景，有的时候也许我们只想知道被索引的数据中是否包含某个限定的条件（类似第 5 章提到的布尔检索模型），这时候，设计者就可以使用过滤查询，它只会判断查询条件是否出现，而不会根据打分公式进行复杂的计算，这样也能提升查询的效率。
- ❑ 限制结果读取的范围。对于大型的搜索引擎，查询一些比较宽泛的关键词或条件时，往往会返回大量的结果，如果要遍历所有的内容，那么耗时将是非常巨大的，很难



实现在线服务。好在通常情况下，我们只需要向用户返回前若干项的结果<sup>①</sup>。但是，不排除有网络爬虫或恶意的黑客逐页地获取结果，导致对搜索系统造成无法预见的压力。这也是目前主流的搜索引擎，会限制用户只能访问前 100 页内容的原因。

关于索引和查询的优化策略，目前都是针对单机纵向的优化。不过，再好的优化策略、再好的硬件性能，都是有其瓶颈的。在大数据时代，是离不开横向的水平扩展的。第 5 章中介绍了 Solr 的分片索引和查询机制。分片后索引的尺寸可以大幅降低，并且可进行副本备份以用于容灾。这里列出图 12-12 和图 12-13 的流程作为简短的复习。

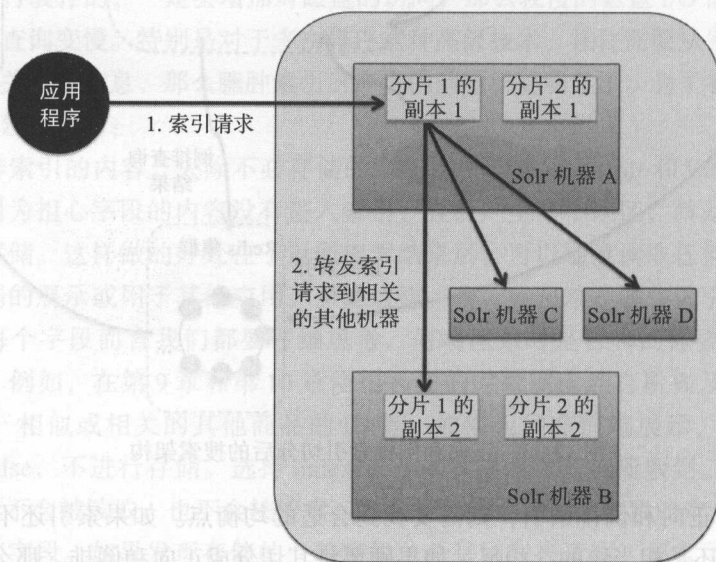


图 12-12 分布式环境中分发索引的更新请求

分布式架构的实现更为精细，好在 Solr 和 Elasticsearch 这样的开源系统都已经提供了比较成熟的方案，对应用开发者而言分布式架构都是透明不可见的，通常不用关心其中的细节。不过，我们还是可以有可以优化的地方，那就是分片的策略。最基础的分片策略是随机式的，根据进来的数据 ID 散列值将其放入到某个分片上。其优势就是实现简单，无须过多的额外开发。不过，对于性能优化而言，建议根据实际应用来确定。例如，大宝公司的 O2O 电商平台，有个“2/8”特征，就是 80% 的流量集中在 20% 的热门品类上。基于此，我们完全可以根据不同的品类来进行切分，这样做的好处是：

❑ 对于类目的查询，只需要访问固定的分片，从而可以尽量避免查询的合并，提升响应的速度。

❑ 对于热门品类，可以有针对性地部署额外的硬件资源。而对于访问量很少的品类，

① 研究表明，绝大部分的用户只会关注前几页的搜索结果。



可以投入更少的硬件资源，以节约成本。

□ 对于搜词的查询，由于其结果经常是跨品类的，因此对查询的合并无法避免。不过，我们有个利器可以有效地降低合并的次数，那就是之前有所提及的词条-类目表，为了便于理解，这里再次列出：

搜索词 相关类目列表

剃须刀 电动剃须刀：0.75，手动剃须刀：0.19，剃须啫喱：0.04，刀片：0.01，……

牛奶 国产牛奶：0.58，进口纯牛奶：0.19，酸奶：0.11，风味奶：0.04，……

咖啡 国产咖啡：0.75，进口咖啡：0.15，咖啡饮料：0.04，进口咖啡豆：0.02，……

手机 手机：0.92，手机保护套：0.02，移动电源：0.01，手机存储卡：0.01，蓝牙耳机：0.01，……

女装 女羽绒服：0.31，女外套：0.23，女连衣裙：0.15，女T恤：0.05，中老年女装：0.03，……

……

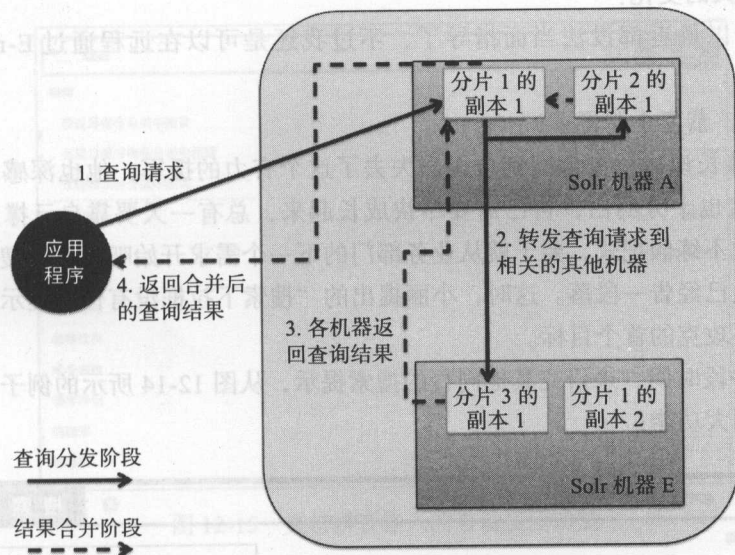


图 12-13 分布式环境中分发查询的请求，并合并查询结果

例如，对于关键词“牛奶”，最相关的分类都是食品饮料，对于关键词“女装”，最相关的分类都是服饰类，因此搜索它们时只需要访问相关品类的分片即可。如此，不仅能提高查询的效率，还能减少不相关商品的返回率，一举两得。

“没想到词条-类目表在这里也发挥了重要的作用啊！”大宝颇有感慨。

“哈哈，词条-类目表的妙用还不止这些呢，后面也许你还会用到哦”。

和正向、倒排索引的切分类似，是否要做分布式的分片和副本，要依实际情况而确定。如果单机的软硬件优化后，性能足够高的话，那么并不建议采用分布式架构，因为这样会导致更复杂的系统架构，延长了查询的周期，并且会产生更高的开发成本和维护成本。

## 12.10 业务需求：给点提示吧

在小明的大力支持下，大宝和自己的团队，针对业务需求进行了架构上的大幅优化，搜索索引数据的更新，以及查询的响应时间都有了明显的改善。为此，大宝特地找到小明进行答谢。

“大宝，何必这么客气。”

“这一路走来，你帮助我太多太多，没有这些宝贵的经验，我和我的团队，包括整个公司，可能几个月之前都已经无法发展下去了，更别提上市的梦想啦！”

“能帮助你们实现创业梦想，我也很高兴。不过，上周刚刚收到老板的消息，根据项目和公司的安排，我预计要去美国深造几年。”

“啊，这么大的变化？”

“是的，所以最近都没法当面指导了，不过我还是可以在远程通过 E-mail 对你进行支持。”

“不管怎样，恭喜小明哥了！”

与小明促膝长谈后，大宝回到公司。失去了这个有力的援军，他也深感肩上的压力重大。不过，大宝也十分明白，自己需要尽快成长起来，总有一天要靠自己撑起公司的一片天。好吧，光说不练假把式，要干就从业务部门的下一个需求开始吧！由于搜索排序的相关性和架构的改良已经告一段落。这时，小丽提出的“搜索下拉框没有任何提示”的需求，就成为了当下需要攻克的首个目标。

大宝花了一段时间，来研究其他同行的搜索提示，从图 12-14 所示的例子可以看出，自动提示主要有两大功能：

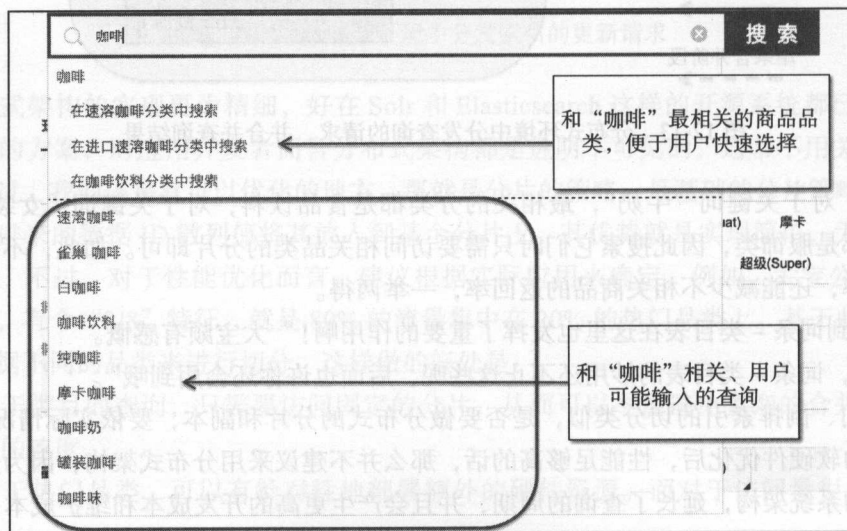


图 12-14 搜索下拉框的自动提示功能

- 相关商品分类：例如，用户输入“咖啡”这个关键词后，和“咖啡”相关的分类都会显示出来，包括“速溶咖啡”、“进口速溶咖啡”和“咖啡饮料”等。大宝觉得这个和小明介绍的重要模块“词条-类目表”是一样的原理，可以直接运用。
- 相关查询：例如，用户输入“咖啡”这个关键词后，“速溶咖啡”、“雀巢咖啡”等用户常常输入的相关查询都会显示出来。这是常用搜索引擎都会提供的功能，不过大宝还真不知道该如何实现，只是依稀记得 Solr 项目中好像有提供类似的功能。

除了图 12-14 所示的主要功能，大宝还发现一些做工精良的网站，能提供更强大的提示功能。例如，图 12-15、图 12-16 分别展示了用户输入拼音和首拼后的提示，图 12-17 展示了用户输入错误拼音后的模糊提示，图 12-18 展示了用户输入繁体中文后的提示。为此，他决定好好研究一番这些功能具体是如何实现的。

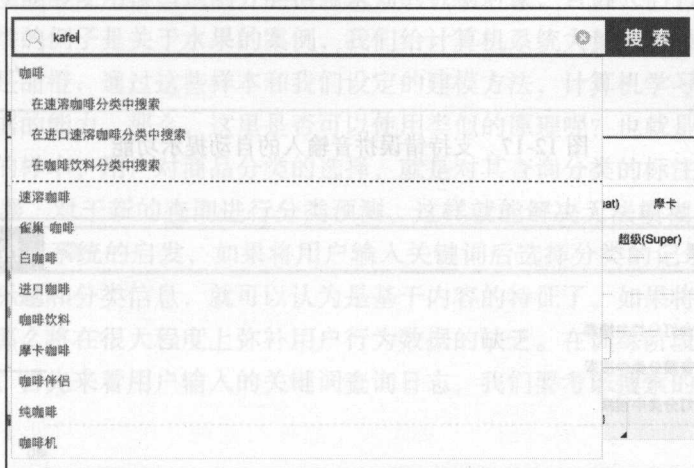


图 12-15 支持拼音输入的自动提示功能

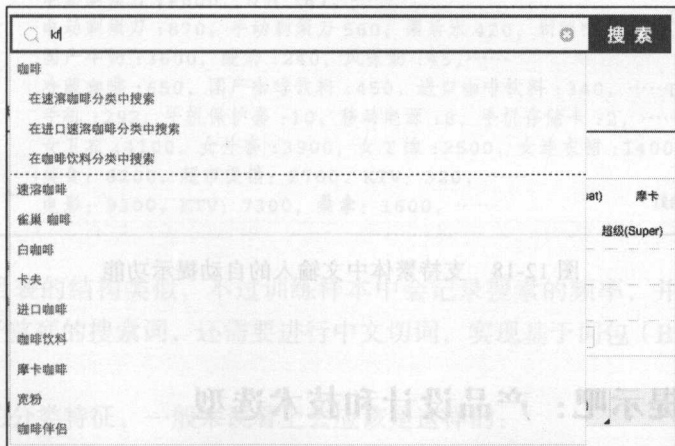


图 12-16 支持首拼输入的自动提示功能

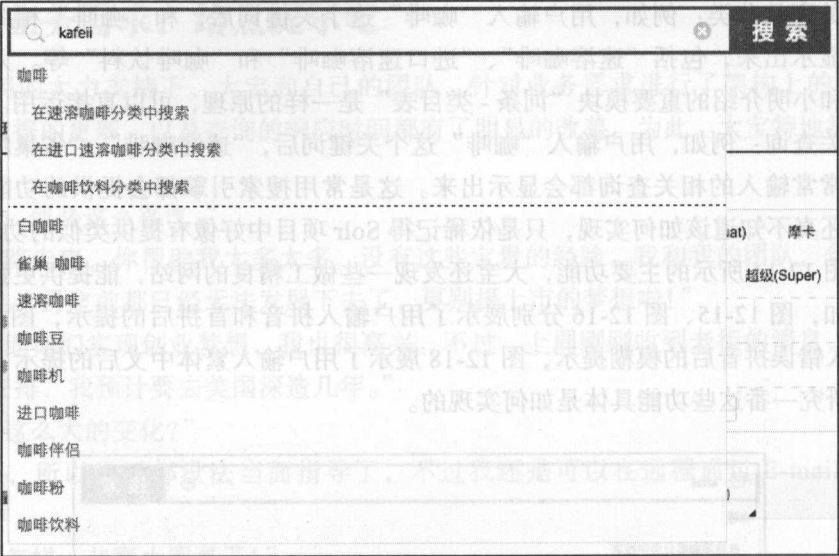


图 12-17 支持错误拼音输入的自动提示功能

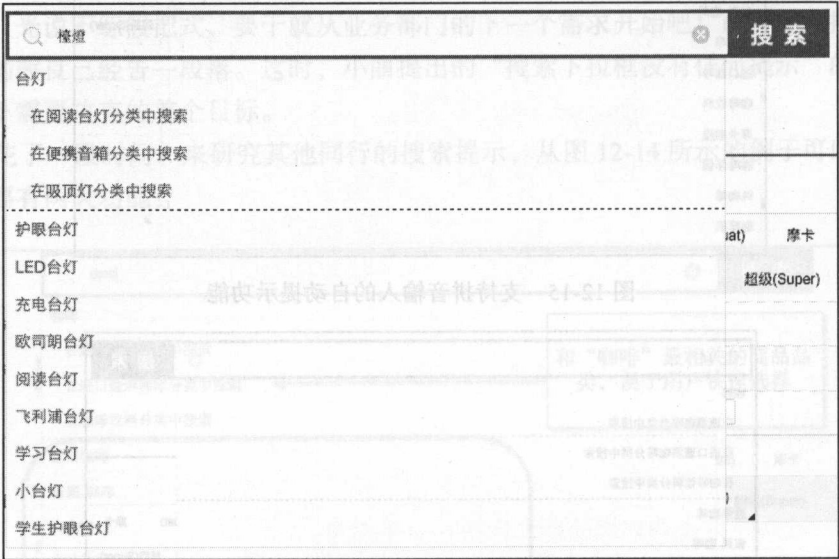


图 12-18 支持繁体中文输入的自动提示功能

## 12.11 给点提示吧：产品设计和选型

根据主流的搜索自动提示功能，大宝及其团队兵分两路开始攻克技术难题。其中一队



主要是利用词条-类目表的数据来对相关的商品分类给出提示。不过，团队成员很快就发现完全依赖用户行为的数据存在以下两点不足。

❑ 无法匹配超长查询。对于一些长度超过普通需求的查询，可能里面的关键词单独拆开来，都有用户的访问记录。但是将它们拼在一起组成的查询，却无法找到对应的用户访问数据，且无法给出分类提示。导致这种情况的原因，主要是词条-类目表里的键-值的查找，只能支持严格的精确匹配，而不支持模糊匹配。

❑ 无法匹配罕见查询。对于一些不常见的查询，由于缺乏用户的访问行为，因此也就无法给出分类提示。导致这种情况的原因，主要是词条-类目表里的数据源还不够丰富。

这时候，大宝想到了第 6 章介绍的分类型技术。用于预测的分类旨在找出描述和区分数据类的模型，以便能够使用模型预测分类信息未知的数据对象，告诉人们它应该属于哪个分类。第 6 章中所举的例子是关于水果的案例，我们给计算机系统大量的水果，然后告诉它哪些是苹果，哪些是甜橙，通过这些样本和我们设定的建模方法，计算机学习并建立模型，最终拥有判断新数据的能力。那么，这里是否可以使用类似的原理呢？也就是说，将用户大量的行为作为训练的样本，用户对商品分类的选择，就是对其查询分类的标注。然后，利用这些标注训练分类器，对于新的查询进行分类预测，这样就能解决无法模糊匹配的问题。此外，大宝还受到推荐系统的启发，如果将用户输入关键词后选择分类的记录作为行为特征，那么商品本身的标题和分类信息，就可以认为是基于内容的特征了。如果将两个特征结合起来构建分类器，那么将在很大程度上弥补用户行为数据的缺乏。在训练阶段，两种特征的使用方式略有不同。首先来看用户输入的关键词查询日志，我们要考虑搜索的频率，大致看起来是这个样子：

搜索词	相关类目列表
剃须刀	电动剃须刀：7500，手动剃须刀：1900，剃须啫喱：400，……
飞利浦剃须刀	电动剃须刀：4800，刀片：67，……
男士剃须	电动剃须刀：870，手动剃须刀：560，须后水：420，剃须啫喱：310，……
光明牛奶	国产牛奶：3600，酸奶：240，风味奶：45，……
咖啡饮料	冷藏咖啡：650，国产咖啡饮料：450，进口咖啡饮料：340，……
黑色老人手机	手机：292，手机保护套：10，移动电源：8，手机存储卡：2，……
韩式女装	女卫衣：4100，女外套：3900，女T恤：2500，女连衣裙：1400，……
优惠套餐	美食：8200，超市促销：2700，KTV：320，……
娱乐	电影：9300，KTV：7300，桑拿：1600，……
……	……

跟词条-类目表的结构类似，不过训练样本中会记录搜索的频率，并用于定制训练的模型。另外，对于首列的搜索词，还需要进行中文切词，实现基于词包（Bag of Word）的分类型模型。

而对于商品的分类特征，一般来说看上去应该是这样的：

商品名称	类目
------	----

喜洋洋牌超锋利超酷炫彩男士电动剃须刀	电动剃须刀
美滋滋牌黑色超帅水洗电动剃须刀 送啫喱膏	电动剃须刀
喜洋洋牌男士调理须后水 150ml/瓶	须后水
美滋滋牌男士爽肤水 200ml 补水保湿须后水	须后水
小母牛牌纯牛奶 250ml*24/箱	国产牛奶
大花牛牌特别纯系列纯牛奶 250ml*12/箱*2	国产牛奶
小母牛牌利乐枕纯牛奶 240ml*12/箱	国产牛奶
大花牛牌全程有机奶精品装 200ml*12盒/箱	国产牛奶
每周必唱 KTV 工作日欢唱 5 小时	KTV
笑哈哈影院电影周二特价场	电影
……	

同样，对于首列的商品名称，也需要进行中文切词。对于分类器的实现，大宝让团队选择了易于理解、实现和调试的朴素贝叶斯模型（Naive Bayes），在 Mahout 和 R 中也都有现成的实现方案。至于行为特征和内容特征的综合，他们也选择了比较灵活的宏观混合方式，将基于用户行为的分类器结果和基于商品内容的分类器结果进行加权平均，获得一个综合的分值用于最后分类的判定。由于缺乏人工的标注，这里的分类效果难以通过离线的交叉验证来评估。好在有大量线上用户的行为可以作为佐证。常见的方式是，通过一些假设，将他们的点击和购买作为参考答案，测试整体的准确率。

另一队成员，则要负责相关查询的提示，他们本想利用 Solr 自带的 Suggester 组件直接完成这项任务的，但是实际操作后他们发现了其中几个不太适合的地方：

- ❑ 默认只支持前缀匹配。这点非常符合英文这种语系的特征，和英文的书写顺序一致。例如在输入“co”的时候，会弹出“coffee”、“coconut”、“computer”等常见查询。但前缀的匹配并不适合中文的拼写习惯。例如输入“咖啡”，如果只关注前缀那么只会提供“咖啡”、“咖啡机”、“咖啡壶”这样的查询，而顾客可能想搜索的是“白咖啡”、“速溶咖啡”、“罐装咖啡”等。
- ❑ 默认只关注字符串的相似度。由于自动提示时，用户还没有完成整个查询的输入，因此字符串的相似度计算是很关键的。不过，这还远远不够，还需要考虑查询被搜索的热门程度和网站商品的匹配程度等。
- ❑ 默认只关注商品的标题。很多时候，设计者在使用 Suggester 时，误将其和商品的索引绑定在一起，因此系统只会对商品的标题进行自动提示。实际上，顾客更为关心的是其他用户的输入，而不是商品标题本身。

这些都会导致自动提示相关查询的效果很差，大宝针对直接使用 Suggester 的不足，再结合业务的需求，和团队成员一起制定了新的技术方案：

首先，他们大胆地将用于搜索提示的索引和商品的索引分隔开来，专门针对用户的搜索日志来进行索引，这就确保了提示内容的质量。其次，极大地丰富了搜索日志中的内容，除了原始的关键词内容以外，还包括了查询的相关分类、繁体、拼音和首拼等，这就使得在查询的时候，不仅可以支持简体中文，还可以支持其繁体和拼音等变体，并且同时返回相关分类的信息。最终，利用 Solr 强大的搜索功能，还可以考虑除了相关性之外的其他因素，

例如搜索次数和匹配商品的数量等。根据这个主题思路，大致的整体架构如图 12-19 所示。

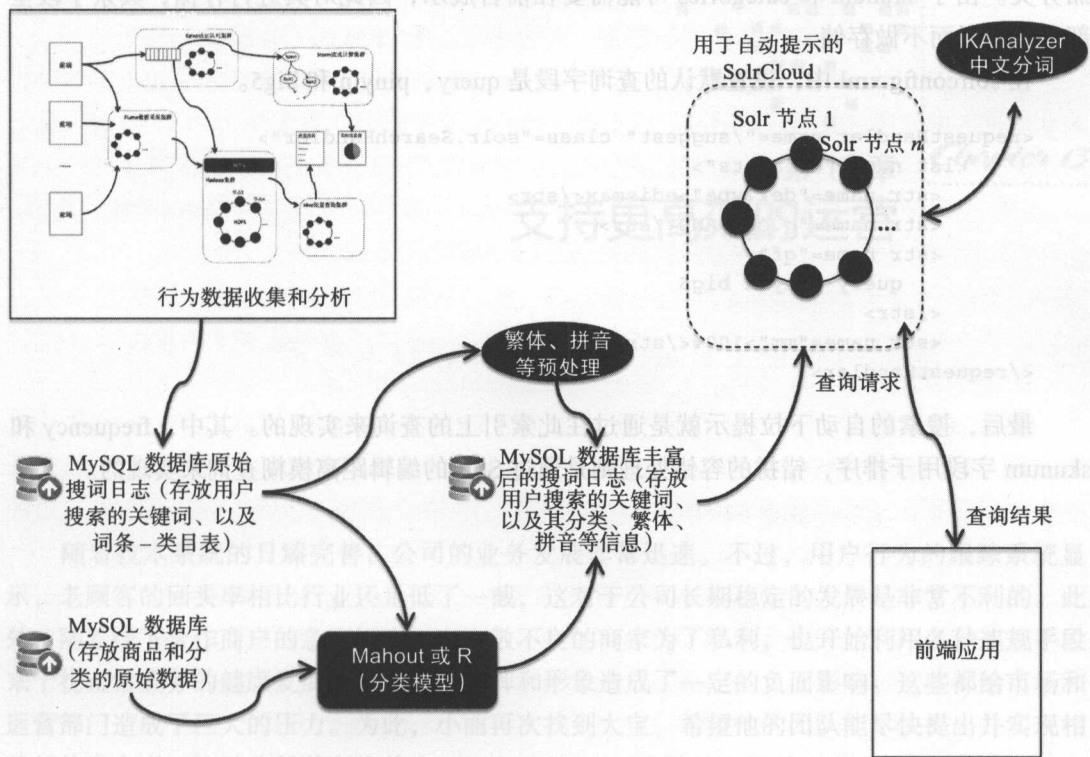


图 12-19 搜索下拉自动提示的技术框架

为了便于行为特征和内容特征的整合，大宝团队并未采用 Mahout 类或 R 来实现朴素贝叶斯的分类，而是自己进行了编码，同时将行为特征的权重设置得更高，从而获得了理想的分类效果。

在 Solr 的 schema.xml 中，进行如下字段的配置：

```
<field name="queryid" type="string" indexed="true" stored="true"
required="true" multiValued="false" />
<field name="query" type="text_chinese" indexed="true" stored="true"
required="true" />
<field name="pinyin" type="text_chinese" indexed="true" stored="false"
required="true" />
<field name="big5" type="text_chinese" indexed="true" stored="false"/>
<field name="frequency" type="int" indexed="true" stored="false"/>
<field name="skunum" type="int" indexed="true" stored="true"/>
<field name="categories" type="text_chinese" indexed="false" stored="true"
multiValued="true" />
```

其中，query、pinyin、big5、frequency、skunum 和 categories 字段分别表示用户查询的

原始输入、拼音（包括首拼）、繁体、搜索频率（表示热门程度）、匹配的商品数量和相关商品分类。由于 skunum 和 categories 可能需要在前台展示，因此对其进行存储，其余字段全部只做索引而不做存储。

在 solrconfig.xml 中，设置默认的查询字段是 query、pinyin 和 big5。

```
<requestHandler name="/suggest" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="defType">edismax</str>
    <str name="q.op">AND</str>
    <str name="qf">
      query pinyin big5
    </str>
    <str name="mm">100%</str>
  </requestHandler>
```

最后，搜索的自动下拉提示就是通过在此索引上的查询来实现的。其中，frequency 和 skunum 字段用于排序，错拼的容错处理则是通过 Solr 的编辑距离模糊查询来实现的。

在 Mahout 和 R 中也有类似的功能，他们选择了比较灵活的宏规则来组合方案。在 Mahout 和 R 中也有类似的功能，他们选择了比较灵活的宏规则来组合方案。

另一队成员，则要负责相关查询的提示。他们本想利用 Solr 自带的 Suggester 组件直接完成这项功能。但是实际操作后他们发现，其中有一些问题需要解决。

首先，默认只支持前缀匹配。这点非常符合英文的习惯，但在中文的场景下，用户输入“co”的时候，会弹出“coffee”、“coco”、“computer”等常见查询。

但前缀的匹配并不符合中文的习惯。例如输入“咖啡”，如果只关注前缀那么只会提供“咖啡”、“咖啡机”、“咖啡壶”这样的查询，而顾客可能想搜索的是“白咖啡”、“黑咖啡”、“咖啡粉”等。

因此字符串的相似度计算是很关键的。不过，这还远远不够，还需要考虑搜索的热门程度和网站商品的数量等因素。

在 Solr 的 schema.xml 中，默认只关注商品的标题。很多时候，设计者在使用 Suggester 时，误将其和商品的索引绑定在一起，因此系统只会对商品的标题进行模糊匹配。

这会导致系统只能返回与商品标题相关的搜索结果，而无法返回与商品描述、属性等相关的搜索结果。因此，在设计 Suggester 时，需要考虑到多个字段的匹配。

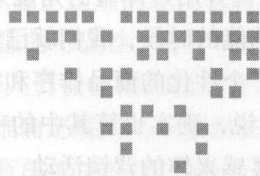
结合业务的需求，和团队成员一起制定了新的技术方案：首先，我们利用 Solr 的 Suggester 组件，对商品的标题进行模糊匹配。

其次，我们利用 Solr 的 Suggester 组件，对商品的描述、属性等进行模糊匹配。最后，我们利用 Solr 的 Suggester 组件，对商品的分类进行模糊匹配。

除了原始的关键词内容以外，还包括了商品的分类、属性、品牌等信息。这使得在查询的时候，不仅可以支持简体中文，还可以支持繁体和拼音等变体。

并且同时返回相关的商品列表。最后，利用 Solr 强大的搜索功能，还可以考虑除了相关性之外的其他因素，如商品的价格、销量等。





## 支持更高效的运营

随着技术系统的日臻完善，公司的业务发展非常迅速。不过，用户行为的跟踪系统显示，老顾客的回头率相比行业还是低了一截，这对于公司长期稳定的发展是非常不利的。此外，随着线下合作商户的急剧增多，有少数不良的商家为了私利，也开始利用各种违规手段来干扰正常业务的健康发展，对公司的口碑和形象造成了一定的负面影响。这些都给市场和运营部门造成了巨大的压力。为此，小丽再次找到大宝，希望他的团队能尽快提出并实现相关的技术方案，协助公司共度难关。

### 13.1 业务需求：互联网时代的 CRM

“大宝，这些需求的迫切性在管理层的会议上已经描述得比较清楚了。我们先看看最为重要的客户吧，你们的用户行为跟踪系统显示，老顾客的回头率相比行业还是低了一截，这对于公司的长期稳定发展是非常不利的。我们怎样才能更好地管理和激励用户，增加顾客的留存率和活跃率呢？尤其是在互联网领域，用户的忠诚度普遍都不高，我们需要马上采取行动。”

“是的，小丽，不过我还需要明确下具体的需求。”

“我们是否要参考传统的客户关系管理 CRM（Customer Relationship Management）的模式。传统的模式主要是利用技术的手段，来帮助我们的业务部门实现市场营销、销售、服务等活动自动化，如此可以更高效地为客户提供满意、周到的服务，充分体现公司以客户为中心的管理理念，相信能明显地提升顾客的忠诚度。不过，在互联网上我想这还真是个不小的挑战：第一，顾客忠诚度较低，很容易被其他竞争对手所吸引；第二，用户不会轻易地泄露自己真实的性别、年龄和职业等关键信息。具体应该怎样实施，我们还没有想得非常清晰。”

“如果是从提升用户体验的角度来出发，其实我们的系统已经有一些尝试了。在之前实现搜索和推荐功能的时候，我们就已经建立了基础的用户画像，利用顾客对于品牌和品类等的偏好，进行了个性化的商品排序和推荐。这是不是也算 CRM 的一种？”

“从广义上说，确实也算其中的一部分。不过，我们还是希望有更多的渠道，例如手机 App 里推送顾客感兴趣的营销活动，或者是发邮件定期唤醒不够活跃的顾客等。此外，我们也希望业务方能够贡献出一份力量，通过良好的运营将 CRM 系统的功效发挥到最大。”

“嗯，完全可以，我们也许可以考虑对用户画像进行扩展，实现更全面的 CRM 功能。”

“看来你已经有思路了，期待技术团队的再一次创新！”

## 13.2 互联网时代的 CRM：产品设计和技术选型

经过进一步的分析，大宝觉得要让互联网的 CRM 成功，必须要将两个大的模块搭建好。第一，用户数据的收集。这里的数据不仅仅是客户在网站上的行为，还可能包括其所处的环境因素，或者是提炼的属性标签等。第二，在用户数据的基础上，进行相关应用的开发。

先看下有哪些用户数据可以收集。用户数据又可以分为原始数据和提炼数据。最基本的原始数据包括网站浏览、购物和致电行为等，这些之前讨论得比较多，这里就不再赘述了。除此以外，还有很多重要的原始信息可以利用。

- 位置：随着移动互联网的兴起，用户的地理位置信息越来越重要。在用户允许的情况下，可以通过全球定位系统 GPS 对其进行定位，或者是通过用户自己选择的商圈、社区来定位。地理位置可以帮助我们了解客户所在区域的群体画像，包括他们的消费习惯、品牌偏好等。
- 气候：利用用户的地理位置，以及第三方的天气预报等资源，获取用户所在地区的天气状况，包括是否晴雨、气温、气压和湿度等。
- 设备：如果允许，可以获知用户的当前设备还安装了哪些其他应用，以便于侧面了解用户的喜好和职业。
- 其他：随着越来越多的智能设备的开始流行，我们甚至还能知道用户所处环境的光亮、磁场、辐射等信息。

除了原始的数据，我们还可以结合人工的运营，生成一些包含语义的用户标签。这里的用户标签，或者说属性标签，是一个具有语义的标签，用于描述一组用户的行为特征。例如，“美食达人”、“数码玩家”、“白领丽人”、“理财专家”等。对于标签的定义，受第 6 章的启发，既可以考虑采用监督式的分类方法，也可以采用非监督式的聚类方法。

- 分类：这种做法的好处在于可以让人工运营向计算机系统输入更多的先验知识，标签的制定和归类更为精准。从操作的层面来考虑，又可以细分为：
  - 通过人工规则指定。例如，运营人员指定最近 1 个月，至少购买过两次以上母婴

产品,消费额在 500 元以上的为“辣妈”<sup>①</sup>标签。这里的规则就相当于直接产生类似决策树的分类模型,它的优势在于具有很强的可读性,便于人们的理解和沟通。但是,如果用户的行为特征过于繁多,运营人员往往很难甄别出哪些更具有代表性。这时如果仍然使用该规则,那么就会很难确定规则的覆盖面或精准度。

- 通过样例来指定。相对于规则而言,更为简单的方法是,通过运营人员挑选一些有代表性的用户,然后输入给系统,让系统根据分类技术来学习,模型可以使用决策树、朴素贝叶斯 NB (Naive Bayes) 或支持向量机 SVM (Support Vector Machine)。虽然在海量标签库里操作起来比较省事,但是对于技术系统的要求会更高。同时,除了决策树的模型,其余模型产生的人群分组会缺乏可读性内容,很难向业务方解释其结果。一种折中的办法是让系统根据数据挖掘中的特征选择技术,包括信息增益 IG (Information Gain)、开方检验 CHI 等,来确定这组人群应该具有怎样的特征,并将其作为标签。

□ 聚类:运营人员参与得最少,完全利用用户直接的相似度来确定,相似度同样可以根据行为特征和内容特征来衡量。其问题也在于结果缺乏解释性,只能通过特征选择等技术来挑选具有代表性的标签。

综合上述两点来看,分类的技术比较适合业务需求明确、运营人员充足、只针对少量高端顾客的管理,其精准性能可以提升贵宾服务的品质。而聚类更适合大规模用户群体的管理,甚至是进行在线的 AB 测试,其对精准性要求不高,但是数据处理的规模有一定的门槛。

利用尽可能多的环境信息,刻画更为完整的用户画像,这是精准化数据营销的根本。有了用户的基础数据,大宝发现互联网模式下的 CRM 其实有很多有趣的玩法:

□ 搜索,投其所好,增加搜索栏位中详情页的点击率和购买转化率。

- 个性化的排序,这点在第 12 章中已经有所阐述,根据用户经常购买的品类和品牌,对搜索结果中的商品进行个性化的排序。
- 个性化的导购属性。根据用户经常购买的品类和品牌,对搜索结果中的导购属性进行个性化的排序。
- 个性化的搜索提示。例如,经常购买儿童洗衣液的用户,输入儿童用品的品牌后,在搜索下拉框中优先提示该品牌的儿童洗衣液。

□ 推荐,基于用户的推荐,这点在第 11 章中已经有所阐述,在用户画像完善的前提下,可以通过包含各种行为和内容特征的数据,找出和当前用户相似的“近邻”。

□ 电子邮件营销 EDM (E-mail Direct Marketing)。传统的线下营销,由于印刷和人力成本的制约,无法做到因人而异的精准化定向投放。而这点在线上却成为了可能。大体上有两种主流的做法:

① 请注意,这里都是根据用户的行为来判定属性标签的,不代表用户的真实性别。



- 人工运营用于营销的电子邮件，根据品类、品牌、节日或时令，分为不同的主题。而对顾客根据其画像，分别和这些主题进行相似度的匹配，相似度计算的 KPI 指标在第 5 章和第 6 章中都有详细的介绍，例如向量的欧氏距离和余弦夹角等。
  - 人工运营并不涉及内容的细节，而是制定一定的模板和规则，然后让系统根据用户画像的特征，自动地填充模板并最终生成电子邮件的内容。
- 移动 App 的推送。随着移动设备逐渐占据互联网市场的主导地位，掌上设备的 App 推送就变成了另一个重要的营销渠道。从技术层面上看，它可以采用和 EDM 类似的解决方案。不过，内容的运营要考虑到移动设备屏幕的尺寸和交互方式的特性，并进行有针对性的优化。
- 与第三方的合作：打通用户的流量也是业界目前最常见的做法。对于 O2O 电商而言，用户的主力军主要是有消费力的年轻人，可以去了解他们关注哪些其他的互联网领域，然后和那些领域的行业领先者进行流量的互换，甚至在某些业务上进行深度合作。这个时候，用户画像就成为双方快速建立沟通渠道的利器，使得跨行业的 CRM 成为可能。

综合这些因素，大宝设计出如图 13-1 所示的整体架构。

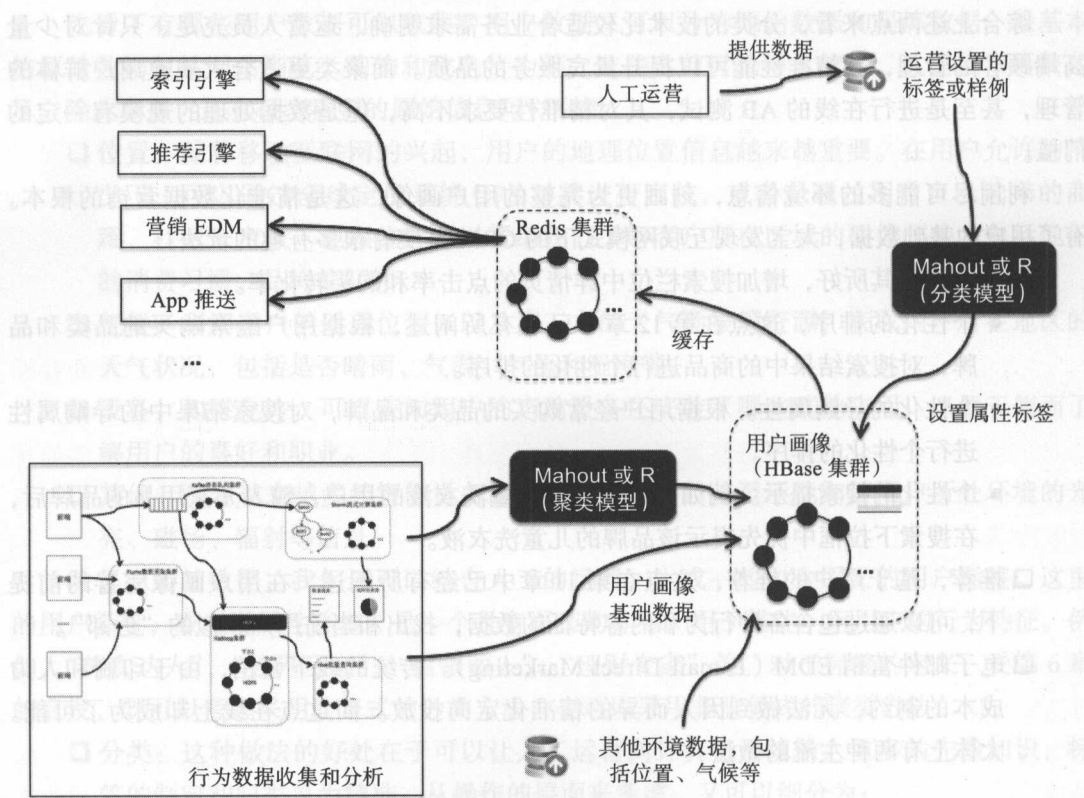


图 13-1 利用用户画像进行 CRM 的管理



这幅架构图是第 12 章中利用用户画像进行个性化搜索排序的扩展。行为数据的收集和分析模块不变, Flume、Kafka、HDFS、Hive 和 Storm 等组件提供用户的基础画像数据。此外, 我们还需要将更多的环境数据写入 HBase 存放, 其间可能需要使用第 2 章介绍过的爬虫机制来获取互联网上的开放资源。Mahout 或 R 可以对行为数据, 或者是人工运营的数据进行聚类、分类的处理, 并最终将属性标签设置到 HBase 里进行存放。HBase 对于异构数据源的包容性、集成性优势再次得以体现。Redis 仍然作为缓存, 提升数据查询的效率, 为前端的搜索、推荐、EDM 和 App 推送等应用提供服务。当然, 我们还可以利用行为数据的跟踪, 进一步分析这套 CRM 系统的质量和效果, 形成一个螺旋式上升的优化闭环。

### 13.3 业务需求: 抓住捣蛋鬼

这天, 大宝和小丽在公司的餐厅不期而遇, 小丽看见大宝主动打起了招呼。

“大宝, 感谢你们开放的 CRM 系统, 我的团队虽然也是刚刚开始上手运营, 不过就目前的使用反馈来看还是相当不错的, 不少业务同仁都说操作很方便, 相对于传统的线下营销手段方便了不少。”

“哪里, 那我们顾客的活跃度有提升吗?”

“CRM 需要的是对用户的深入理解和长期培养, 目前还没看到立竿见影的效果, 不过有了你们的技术支持, 我们很有信心深耕细作。”

“那就好。既然如此, 你最近的压力应该小了不少啊, 可我看你好像还是愁眉不展啊。”

“哎…最近出现了不少捣蛋鬼。”

“捣蛋鬼?” 大宝有些疑惑。

“是啊, 我女儿每天晚上都会看一部动画片, 叫《爱探险的朵拉》。每一集主人公朵拉都会和布茨等朋友们展开一段精彩纷呈的探险旅程, 我女儿是这部动画片的绝对粉丝。不过, 她很讨厌其中的一个反派角色, 叫‘捣蛋鬼’狐狸。每次这个狐狸一出来, 她都会和动画里的人物一起大喊‘捣蛋鬼, 别捣蛋!’。而我们, 现在也碰到了捣蛋鬼的角色。”

“谁是你所说的捣蛋鬼呢?”

“你也知道, 最近加盟了不少新的商家。业绩是上去了, 可是商家一多, 竞争也随之而来。有些不遵守规矩的商家, 总是有意无意地采用一些不正当的竞争手段。最常见的有两种: 第一, 将商品放入不合适的类目, 骗取更高的曝光率。比如, 我们网站进口牛奶卖得特别火, 每天都有几十万人次来浏览这个类目的商品, 那么有些卖面包机的商家就将其所售的面包机也放入进口牛奶的类目, 期望获得更多的用户流量。第二种和第一种类似, 不过其作弊手法不是针对品类, 而是针对商品的标题, 他们利用一些不合理的关键词 SEO, 借以获得更多的搜词流量。举个例子, 一款国产手机的标题里, 写上了‘比苹果 iPhone 手机质量更好’的字眼, 其商家就是希望用户在搜索‘苹果手机’, 或者是‘iPhone’的时候, 他的手机也能被搜索到。”

“哦，原来如此啊！”

“如果是公司内部的项目运营或是人员管理，那都好办。可是，这些外部加盟的商家数量实在太多，我们也没有足够的人手去监管，只能发现一起查处一起。但是我心里很明白，我们所处理的只是冰山一角，还有不少问题还来不及发现。”

“我感觉有些问题是可以技术来帮助你们解决的，容我仔细考虑下。”

“哦，真的？技术可以帮忙？那太棒了，拜托拜托啦！”

### 13.4 抓住捣蛋鬼：产品设计和技术选型

#### 13.4.1 识别分类错放

听到要识别错放类目的需求，大宝第一时间就想到了在做搜索自动提示的时候曾经做过用户查询的分类，这与识别错放类目的原理应该是一致的，只是当时是对用户的查询进行分类，而这里是对商家上传的商品标题进行分类。应该只需要同时使用基于商品类目的内容特征和用户搜词后的行为特征即可。训练数据集沿用第 12 章的例子，举例如下。

基于商品内容特征的训练集：

商品名称	类目
喜洋洋牌超锋利超酷炫彩男士电动剃须刀	电动剃须刀
美滋滋牌黑色超帅水洗电动剃须刀 送啫喱膏	电动剃须刀
喜洋洋牌男士调理须后水 150ml/ 瓶	须后水
美滋滋牌男士爽肤水 200ml 补水保湿须后水	须后水
小母牛牌纯牛奶 250ml*24/ 箱	国产牛奶
大花牛牌特别纯系列纯牛奶 250ml*12/ 箱*2	国产牛奶
小母牛牌利乐枕纯牛奶 240ml*12/ 箱	国产牛奶
大花牛牌全程有机奶精品装 200ml*12 盒 / 箱	国产牛奶
每周必唱 KTV 工作日欢唱 5 小时	KTV
笑哈哈影院电影周二特价场	电影
.....	

基于用户行为特征的训练集：

搜索词	相关类目列表
剃须刀	电动剃须刀 :7500, 手动剃须刀 :1900, 剃须啫喱 :400, .....
飞利浦剃须刀	电动剃须刀 :4800, 刀片 :67, .....
男士剃须	电动剃须刀 :870, 手动剃须刀 560, 须后水 420, 剃须啫喱 310, .....
光明牛奶	国产牛奶 :3600, 酸奶 :240, 风味奶 :45, .....
咖啡饮料	冷藏咖啡 :650, 国产咖啡饮料 :450, 进口咖啡饮料 :340, .....
黑色老人手机	手机 :292, 手机保护套 :10, 移动电源 :8, 手机存储卡 :2, .....
韩式女装	女卫衣 : 4100, 女外套 :3900, 女 T 恤 :2500, 女连衣裙 : 1400, .....
优惠套餐	美食 :8200, 超市促销: 2700, KTV: 320, .....
娱乐	电影: 9300, KTV: 7300, 桑拿: 1600, .....
.....	

图 13-1 利用用户行为进行 CRM 的管理

实践中大宝还积累了一些宝贵经验：

□ 由于可能存在人为作弊的行为，数据里难免有一些干扰因素。通常需要假设：

- 现有商品的分类信息绝大部分都是正确的。有了这个假设，设计者可以忽略基于内容特征来分类的结果误差。
- 用户通过购买行为来作弊的成本更高。有了这个假设，可以为用户行为特征中的搜词购买行为设置更高的权重，避免作弊者通过浏览点击来作弊。

□ 层次型类目导致训练样本数过少。一般商品的类目都是分为多个层级的，如图 13-2 所示，一级类目“食品、饮料、酒水”，二级类目是“糖果巧克力”，三级类目是“润喉糖”。越是细分的类目，其包含的商品数量可能就会越少。这也许会导致训练样本数量不够，分类精度很差的问题。这种情形下，不建议对于过小的分类进行训练和预测。可以考虑对上级类目进行处理。

□ 新的分类导致训练样本数过少。随着业务的不断扩张，有时需要新增商品的分类。可是新分类建立伊始，其中可能都没有已经发布的商品，那该如何进行训练呢？可以考虑利用爬虫技术，获取其他第三方的信息资源，扩充自己训练的数据库。

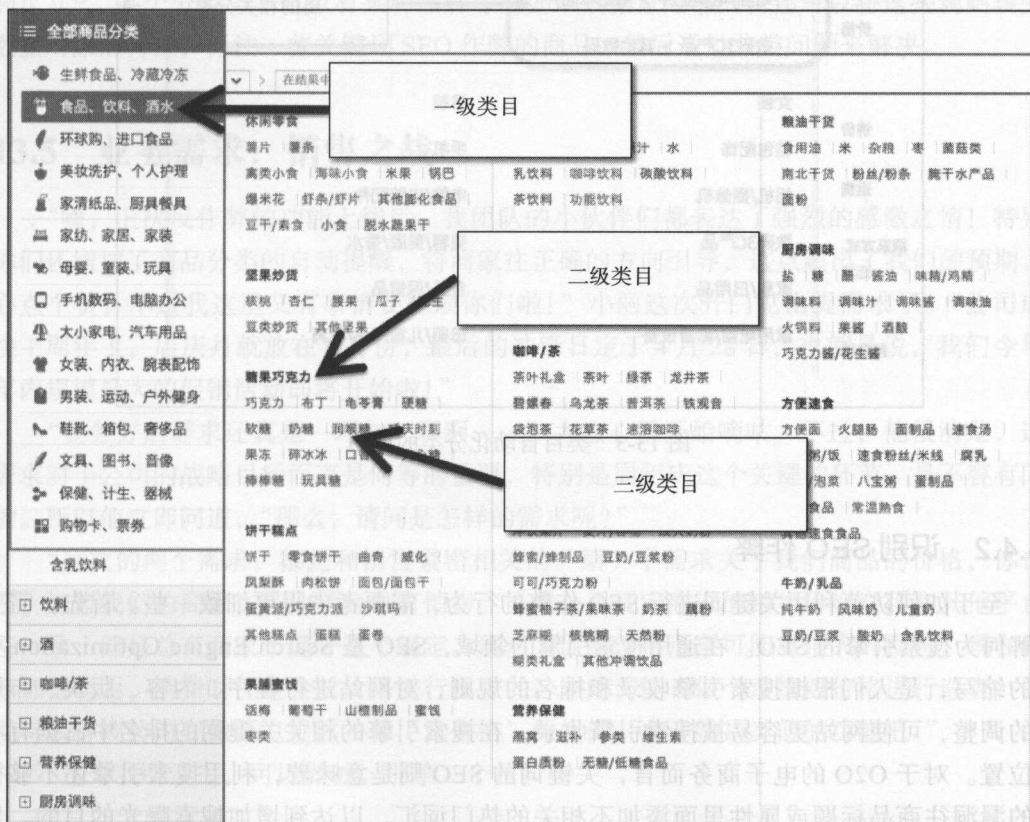


图 13-2 多级类目结构示意图



有了这些想法，大宝的技术团队很快开发出一个功能，在商家刊登商品的时候，自动为其推荐合适的商品分类，其界面如图 13-3 所示。如果商家希望出售一台苹果的 Mac Pro 笔记本电脑，输入“MacBook Pro”的字眼后，系统自动提示最为相关的三个分类“笔记本电脑”、“笔记本配件”和“其他数码”。很幸运，这里算法推荐的第一个分类就是正确的，商家只需要点击选择即可。这样，既方便了商家的商品刊登，又避免了粗心大意而导致的错误分类。而对于企图违规操作的商家，如果选择了与系统默认推荐相差甚远的分类选项，其行为也会被系统记录在案，然后定期生成报表，提交给运营部门进行核查。如此一来，核查人员就可以专注在系统报告的这些嫌疑案例上了，而不必对每项商品逐一审核。自动化的分类算法帮助我们大大降低了全面排查所需的工作量。

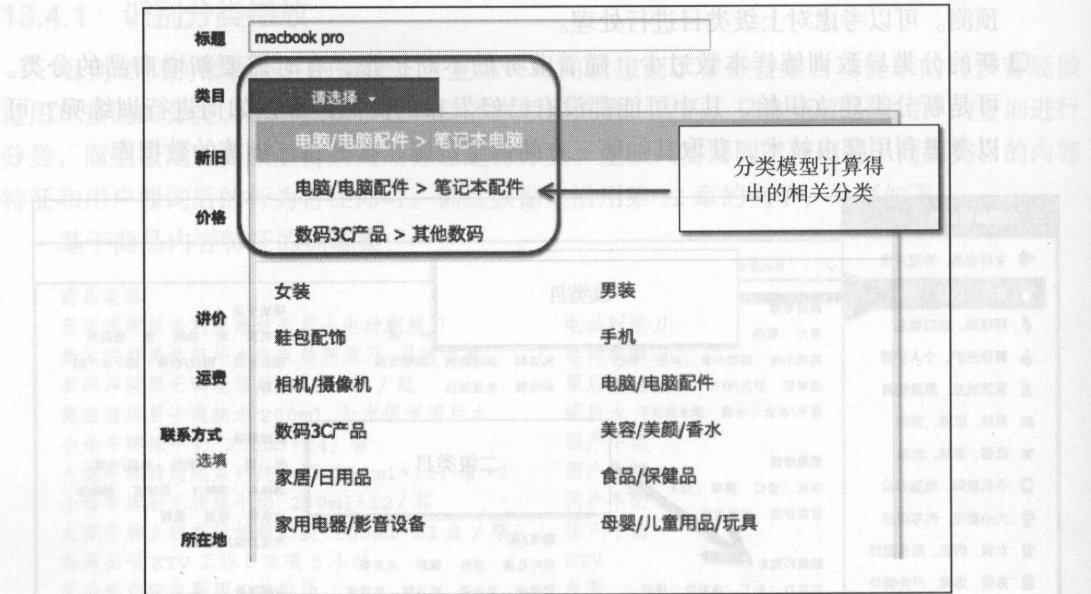


图 13-3 类目自动化分类的应用

13.4.2 识别 SEO 作弊

至于如何防范利用关键词进行 SEO 作弊的行为，需要考虑得更细致一些。首先，需要理解何为搜索引擎的 SEO。在通用搜索引擎的领域，SEO 是 Search Engine Optimization 英文的缩写，是人们根据搜索引擎收录和排名的规则，对网站进行程序、内容、版块、布局等的调整，可使网站更容易被搜索引擎收录，在搜索引擎的相关关键词的排名中占据有利的位置。对于 O2O 的电子商务而言，关键词的 SEO 则是意味着，利用搜索引擎还不够智能的漏洞往商品标题或属性里面添加不相关的热门词汇，以达到增加搜索曝光的目的。例如，之前提及的国产手机的标题里写上“比苹果 iPhone 手机质量更好”的内容，就是一个



经典的案例。

为了识别这种作弊方式，首先需要知道标题或属性中都有哪些关键信息，例如哪些是表明品类信息的，哪些是表明品牌信息的，哪些是表明型号信息的，等等。大宝借鉴了 IK 等中文分词模块的经验，决定使用领域字典来完成。例如，品类字典中就包括“牛奶”、“咖啡”、“手机”，而品牌字段中就包括“A 品牌”、“B 品牌”、“C 品牌”等。不过，值得注意的是，有些词是有很大的歧义的，例如“红豆”、“风筝”这些词既可以是品牌也可以是品类。好在由于商标法的规定，这样的歧义词可以通过不同的商品行业来区分。也就是说，“红豆”作为品类词一定是在食品的杂粮分类中，而作为品牌词一定是在服饰分类中。而“风筝”作为品类词一定是在运动器材分类中，作为品牌词一定是在食品的面粉分类中。最后，这类字典的维护和更新非常重要，因此大宝邀请了小丽的团队进行专项的维护，以确保最终的准确性。

有了分类和字典信息，我们就可以扩展 IK 等中文分词模块了，这样不仅能提高分词的准确率，还能提供品类、品牌、型号、口味、尺寸、颜色等更多的属性。基于这些，就能定义一些防止作弊的规则。例如，一个商品不可以有多个品牌<sup>①</sup>，一个商品不可以有多种型号和尺寸<sup>②</sup>，某个分类下不可以有某种品牌等。如果数据量足够大，还可以将这些规则转变为数据对象的特征，标注一些关键词 SEO 作弊的商品，然后通过分类问题来解决。

## 13.5 业务需求：销售之战

“嗨，上次反作弊的功能上线后，我团队的小伙伴们都表达了强烈的感激之情！特别是你们还增加了商品分类的自动提醒，将商家往正确的方向引导，这点超出了我们的预期，真心点个赞！不过我这里又有事情要麻烦你们啦！”小丽这次开门见山提需求了。“公司成立快 1 周年了，店庆月就放在 4 月份，最后的冲锋日定于 4 月 28 日，也就是说，我们今年一年内规模最大的促销活动即将开始啦！”

“业务方的需求还真是一环扣一环啊……”大宝心里开始嘀咕。不过，他很清楚，这些需求对于公司的战略目标而言是何等的重要，特别是周年庆这个关键的环节，是不容有闪失的。所以他立即问道，“那么，请问是怎样的需求呢？”

“这次的两个需求，都是和销售紧密相关的。第一个需求关乎我们商品的价格，你也知道，加盟我们的商家在其他平台也有销售。虽然公司规定商家的销售价格必须和其他平台保持一致，但是总有商家不遵守这个规定。这对大促的价格形象可能有很大的损害，现在的顾客都很聪明，消费时一定会对价格进行比较的。因此，能否做个系统对价格进行巡查？我听说有些科技公司好像可以提供一种‘比价格’的软件，也许就是类似的功能吧。”

“这个我听说过，可以做的。那第二个是？”

① 有些商品会有主品牌和子品牌，这种情况还需要建立它们之间的关联。

② 服饰的不同颜色，或者数码产品的不同容量等系列品可以除外。

“第二个是关乎黄牛交易的。线下的黄牛大家都很熟悉了，演唱会门票、体育比赛门票，甚至是炒外汇都少不了他们的身影。这不，黄牛也与时俱进，发展到线上来，开始打电商的主意了。只要我们市场活动一推广，发些优惠券，做些低价特惠品，他们一定会来扫货。虽然我们有限制一位顾客的账户只能下一张订单，但是听说他们可以用机器自动注册多个账户来扫货，不知是真是假。如果万一是真的，那我们的优惠券没能落到真正的顾客手里，反而都被这些黄牛给垄断了。这样对公司长期的健康发展非常不利啊！”

“很不幸，这确实是可能存在的。让我想想看，如何解决这两个棘手的问题。”

## 13.6 销售之战：产品设计和技术选型

### 13.6.1 设置合理的价格

要明确一个商品在市场上的价格，不是一件容易的事情。特别是一些非标准化的商品，本身定价就是模糊的。所以，大宝决定聚焦于市面上可以找到的标准化商品。有了这个前提，他将比价系统的核心模块划分为如下几个方面。

#### （1）数据获取

收集商品在其他网站上的销售价格。利用爬虫的技术来获取第三方的数据是必不可少的组件。

#### （2）商品匹配

不同的商品之间，在价格上是完全没有可比性的。因此需要采用一定的技术将外部获取的商品和自家商品进行匹配，找出哪些描述的是同一个商品。这里，商品的品类、品牌、型号、尺寸等物品属性将再次发挥重大的作用。好在技术和运营团队在之前的 SEO 反作弊项目中，积累了良好的字典数据，因此上手非常快，再加上人为规则的设定，要分辨和匹配标准化商品并非难事。

#### （3）定价策略

最基础的策略，就是价格一定要比别人低。不过，大宝和小丽也非常清楚，从公司盈利的角度出发，比价并不一定就是要将自家的价格压到最低。相反，如果发现竞争对手的价格比我们高出不少，还可以考虑让商家适当提价，只要不超出行业标准就行。这样不仅能在顾客心中树立良好的价格形象，还能增加公司的营收，一举多得。在实施的过程中，还有人提出了更佳的建议，让系统同时考虑价格和佣金返点。

这里要介绍下何为佣金返点。第 1 章中提到大宝公司的 O2O 业务模式，是让线下的商家入驻，将其商品或服务列在其平台上。如果有顾客购买，那么根据消费的金额和品类，商家给平台一定的佣金，这也是整个公司的盈利模式。因为不同的品类有不同的毛利，因此佣金返回的百分比点数也不同。例如，高毛利的进口食品，其佣金返点可以达到 8%，而普通国产食品的佣金返点就只有 3%。基于整个特点，我们可以考虑用图 13-4 所示的二维图。最

为理想的状况自然是第二象限，售价比竞争对手低，同时佣金返点高。最糟糕的情况就是售价在行业内非常高，但是佣金却收得很少，既损失了价格形象，又没能获得很好的盈利。

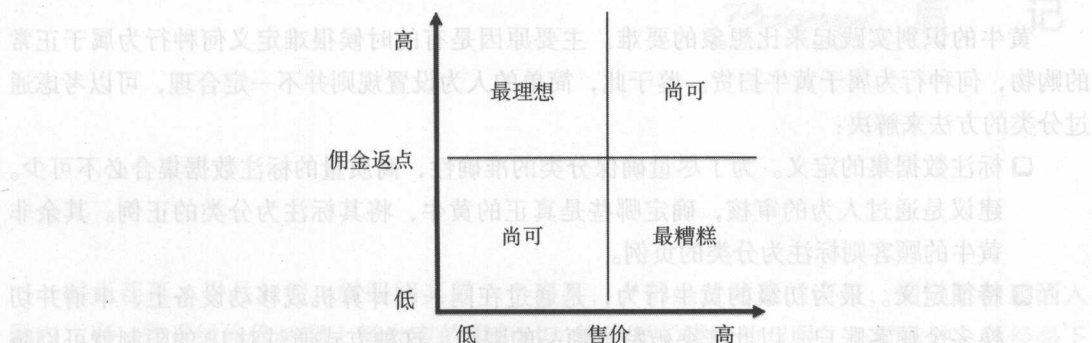


图 13-4 售价和佣金的关系

基于所有这些模块分析，大宝列出了大致的架构设计，如图 13-5 所示。

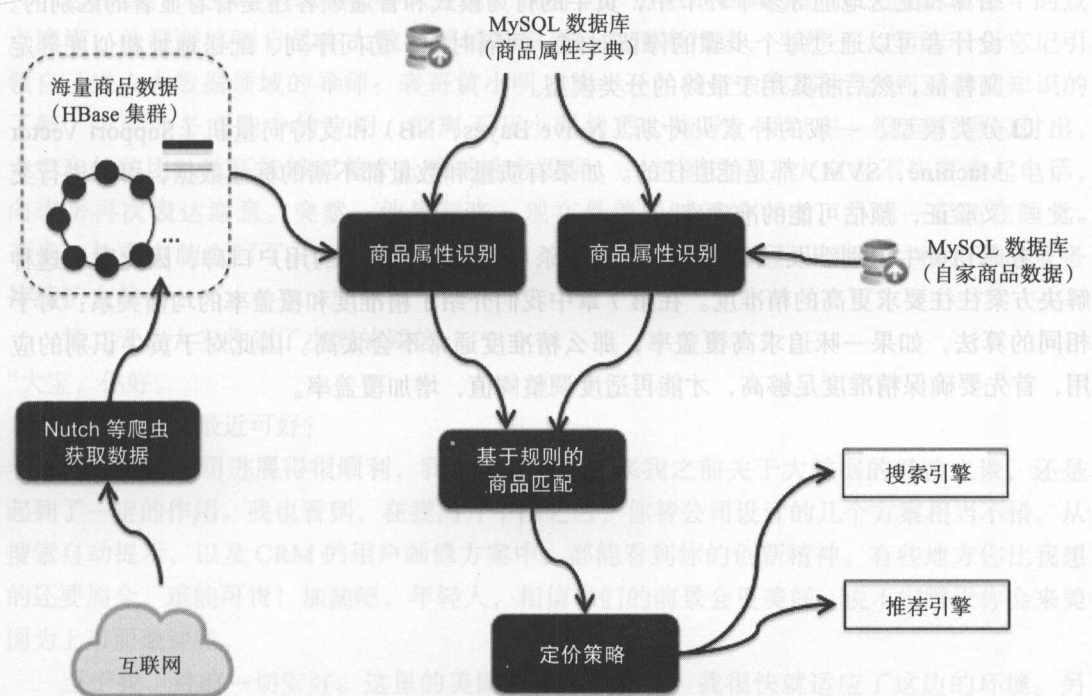


图 13-5 比较系统的框架

从图 13-5 中可以看出，大宝还创新地将自主研发的比价系统和定价策略，运用于网站的其他核心模块。例如搜索和推荐系统中，优先排列符合第二象限特征的商品，希望借此来

提升公司整体的盈利能力，同时赢得顾客关于价格的口碑。

### 13.6.2 识别黄牛

黄牛的识别实践起来比想象的要难。主要原因是有的时候很难定义何种行为属于正常的购物，何种行为属于黄牛扫货。鉴于此，简单的人为设置规则并不一定合理，可以考虑通过分类的方法来解决：

□ 标注数据集的定义。为了尽量确保分类的准确性，高质量的标注数据集必不可少。

建议是通过人为的审核，确定哪些是真正的黄牛，将其标注为分类的正例。其余非黄牛的顾客则标注为分类的负例。

□ 特征定义。最为初级的黄牛行为，是通过在同一台计算机或移动设备上，申请并切换多个顾客账户，以此来突破特惠商品的限购。这种方式通过对 IP 的限制就可以防范和识别。更为精明的黄牛，会采用类似 DDoS (Distributed Denial of Service) 的攻击方式，通过不同的机器 IP 来访问，造成不同的顾客前来购买的假象。但是，他们仍然会露出蛛丝马迹。在用户注册、商品详情页访问、加入购物车，直至最后的结算和配送地址等多个环节中，黄牛的行为模式和普通顾客还是有着显著的区别的。设计者可以通过每个步骤的停留时间、间隔时间、访问序列、配送地址相似度等定义特征，然后将其用于最终的分类模型。

□ 分类模型。一般的朴素贝叶斯 (Naive Bayes, NB) 和支持向量机 (Support Vector Machine, SVM) 都是能胜任的。如果有质量和数量都不错的标注数据，可以进行交叉验证，预估可能的准确率。

考虑到黄牛识别涉及顾客体验，如果错杀可能会导致不良的用户口碑，因此对于这种解决方案往往要求更高的精准度。在第 7 章中我们介绍了精准度和覆盖率的均衡关系：对于相同的算法，如果一味追求高覆盖率，那么精准度通常不会太高。因此对于黄牛识别的应用，首先要确保精准度足够高，才能再适度调整阈值，增加覆盖率。



## Postscript 后 记

时光荏苒，岁月如梭，转眼间又是一个春天，夕阳透过薄薄的窗帘，懒懒散散地洒入屋内。当一缕光线偷偷地爬上杨大宝的眼角时，他睁开了朦胧的双眼。一看挂钟，已经是下午5点钟了。创业一年多以来，他和他的团队就没有好好休息过。在今天凌晨公司大促圆满结束，他也第一次申请了休假，一方面是想调整身心，另一方面也是想对自己创业的一年进行总结。这一觉，大宝睡得很踏实。

起床后，大宝冲了一杯咖啡，斜坐在躺椅上，悠悠地回顾了工作、生活和奋斗的点点滴滴。他深深感到自己在大数据领域的积累已在逐渐增多。当然，大宝绝不会忘记引领自己进入大数据领域的导师：表哥黄小明。从一开始的“大数据盲”，到对基础知识的了解，再到业务实践中的应用，都离不开小明的悉心指导。可以说，没有小明的付出，大宝和他的团队就不能撑起整个公司的业务发展。一想到这里，大宝忍不住要拿起电话，向表哥再次表达谢意。突然，他想起来，现在是美国时间的半夜，小明一定还在睡觉。于是，他在电脑上写了一封长长的感谢信，告诉小明他是如何学以致用，将公司的业务持续扩大的。

第二天，大宝收到了小明的回信：

“大宝，你好。

好久不见，最近可好？

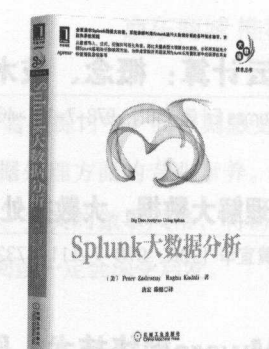
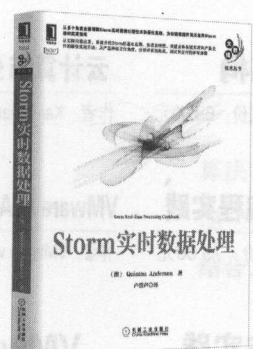
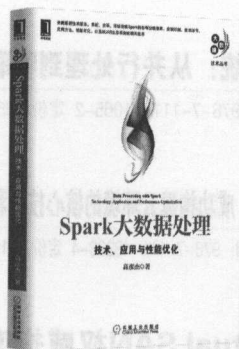
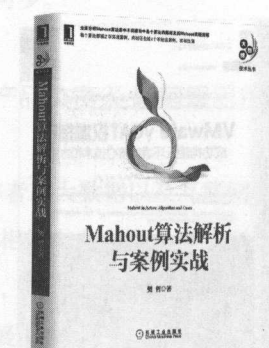
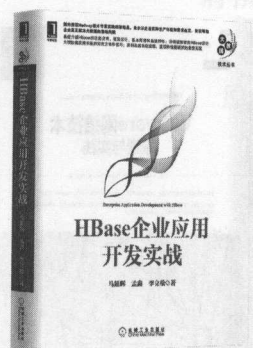
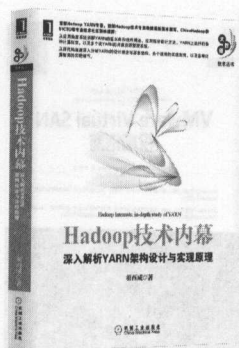
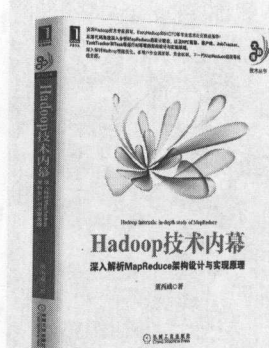
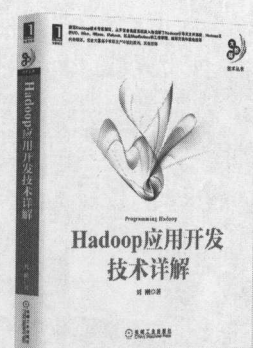
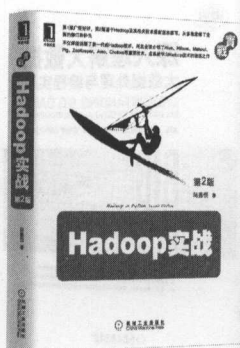
听说你们公司进展得很顺利，我也很高兴。看来我之前关于大数据的经验之谈，还是起到了一定的作用。我也看到，在我离开中国之后，你替公司设计的几个方案相当不错。从搜索自动提示，以及CRM的用户画像方案中，都能看到你的创新精神。有些地方你比我的还要周全，难能可贵！加油吧，年轻人，相信你们的前景会更美好，说不定哪天你会来美国为上市而敲钟呢。

至于我，目前一切安好。这里的美国同事都很友善，我很快就适应了这边的环境。另外，我也发现了很多技术上和生活上有意思的地方，有机会我再和你当面聊聊。我想在美国多深造几年，学成之后，再回到祖国，毕竟中国互联网的机会还很多，希望到时候我们还有合作的机会！”

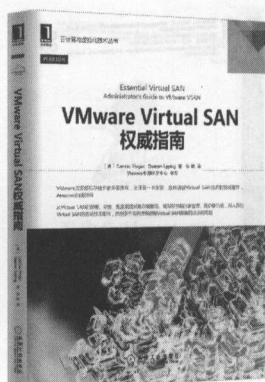
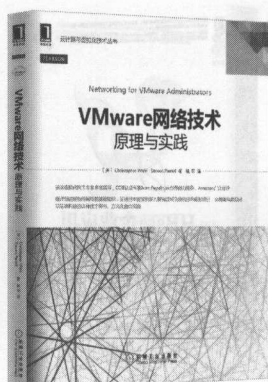
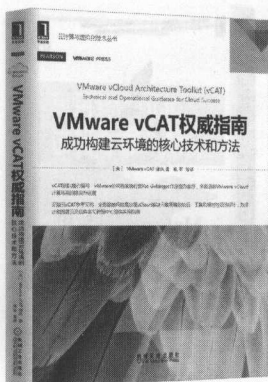
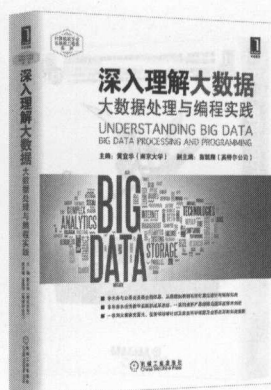
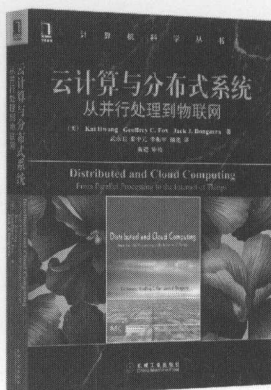
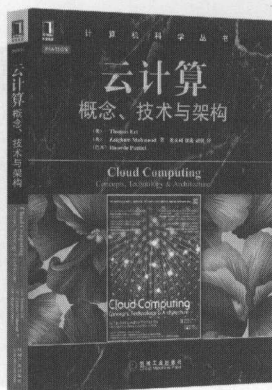
大宝看完微微一笑，回复了5个字：“等你好消息！”



## 推荐阅读



# 推荐阅读



## 云计算：概念、技术与架构

作者：Thomas Erl 等 ISBN：978-7-111-46134-0 定价：69.00元

## 云计算与分布式系统：从并行处理到物联网

作者：Kai Hwang 等 ISBN：978-7-111-41065-2 定价：85.00元

## 深入理解大数据：大数据处理与编程实践

作者：黄宜华 ISBN：978-7-111-47325-1 定价：79.00元

## VMware vCAT权威指南：成功构建云环境的核心技术和方法

作者：VMware vCAT 团队 ISBN：978-7-111-48228-4 定价：119.00元

## VMware网络技术：原理与实践

作者：Christopher Wahl 等 ISBN：978-7-111-47987-1 定价：59.00元

## VMware Virtual SAN权威指南

作者：Cormac Hogan 等 ISBN：978-7-111-48023-5 定价：59.00元



作者对业务运营具有深刻理解，他加盟1号店的阶段，公司在搜索、数据处理技术方面的实力猛增，相关体验和口碑大幅提升。这次他将其宝贵的实战经验在此书中和大家分享，相信对于广大读者而言实在是非常棒的福利，不容错过。

——黄志雄

原1号店副总裁 现永辉集团电商总经理

我和作者有过不少项目合作，其敏锐的业务洞察力给我留下了深刻的印象。从他提供的书稿中，我确实体会到其深厚的专业功力和精心的全文构思。对于每位大数据产品经理而言，这样深入浅出的书籍必不可少。

——张旭强

原1号店产品负责人 现阿里巴巴高级产品专家

本书作者对大数据以及互联网技术有着自己独特的见解。相信本书会给大家带来更为完整和详细的技术剖析，帮助读者更好地理解技术如何支撑业务的高速发展。

——刘尚堃

京东商城 推荐搜索部总监

在与作者的探讨中，我深刻感受到了他在挖掘算法和大数据处理方面的专业素养。这本书秉承了他一贯严谨、务实的做事风格，将需求和技术紧密结合，仔细阅读一定会深受启发。

——诸超

唯品会 云计算高级总监

作者在1号店的三年（2012-2014年），正是1号店系统快速发展的三年，即从大型电商系统转向巨型电商系统的阶段。而作者在此期间的贡献，对1号店搜索系统的变革起到了关键的作用，他和他的团队也藉此获得了“总裁特别奖”。通读全书，我发现该书不仅是大数据技术的探讨，也是技术和业务结合的心路历程。对大数据感兴趣的同行，定能从中获得全新的认识。

—— 韩军 原1号店CTO 现欧电云科技董事长

我和作者曾经一起负责过eBay全球的数据挖掘项目，他的商业敏感度和创新精神让我记忆犹新。不过没有想到，对于写书，他也是个好手。此书既讲述大数据的理论知识，也介绍实际经验，适合不同层次的读者，并能帮助他们解决商业应用中的困惑与难题。

—— Yongzheng Zhang LinkedIn（领英）商务分析经理

我曾经负责整个1号商城的运营，同作者有过不少项目上的对接。对于我们提出的运营相关问题，作者都能运用合适的技术方案，顺利地解决。当得知他要撰写一本与大数据技术相关的图书，我已经迫不及待地想一探究竟，感受业务和技术融合的神奇。希望广大读者在读完此书后也能收获颇丰。

—— 吴海泉 原1号店副总裁 现美的集团电商总经理

作为互联网企业，通常面临的难题是大数据相关的产品设计门槛较高，缺乏专业的人才。当作者向我介绍这本书的时候，我没有想到他竟能将种种复杂的技术问题，说得如此生动、易懂、易学。相信这样的书籍，对数据产品经理的培养、大数据技术的产品化都有不小的促进作用。

—— 王欣磊 百度LBS新业务产品总监



投稿热线: (010) 88379604  
客服热线: (010) 88379426 88361066  
购书热线: (010) 68326294 88379649 68995259

华章网站: [www.hzbook.com](http://www.hzbook.com)  
网上购书: [www.china-pub.com](http://www.china-pub.com)  
数字阅读: [www.hzmedia.com.cn](http://www.hzmedia.com.cn)

